UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

TAGARELA 2.0: FRAMEWORK DE COMUNICAÇÃO ALTERNATIVA, MÓDULO DE JOGOS

ELVIS MERTEN MARQUES

BLUMENAU
2014

2014/2-07

ELVIS MERTEN MARQUES

TAGARELA 2.0: FRAMEWORK DE COMUNICAÇÃO ALTERNATIVA, MÓDULO DE JOGOS

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

Prof. Dalton Solano dos Reis, M.Sc.- Orientador

BLUMENAU
2014

2014/2-07

TAGARELA 2.0: FRAMEWORK DE COMUNICAÇÃO ALTERNATIVA, MÓDULO DE JOGOS

Por

ELVIS MERTEN MARQUES

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

Presidente:	Prof. Dalton Solano dos Reis, M.Sc – Orientador, FURB
Membro:	Prof. Alexander Roberto Valdameri, M.Sc – FURB
Membro:	Prof. Aurélio Faustino Hoppe, M.Sc – FURB

Blumenau, dia 11 de dezembro de 2014

Dedico esse trabalho a todos que me ajudaram a entregar essa monografia a tempo.

AGRADECIMENTOS

Agradeço primeiramente ao Google por sua gama de fontes para pesquisa gratuita.

A minha família por ajudar no pagamento da mensalidade da FURB.

As desenvolvedoras de games que não produziram nenhum jogo de meu interesse durante o semestre e me permitiram dedicar mais tempo na produção desse trabalho. Principalmente a Rockstar Games por fazer o lançamento de GTA V para PC apenas no início do próximo ano.

A Ana Leticia Travaglia Stray, minha luz de Eärendil, minha estrela mais amada. Que me iluminou quando todas as outras luzes se apagaram.

Ao professor Dalton Solano dos Reis por me auxiliar principalmente nesse último ano.

Stay Hungry. Stay Foolish.

The Whole Earth Catalog

RESUMO

Este trabalho é uma extensão do projeto “Aplicativo para comunicação alternativa no iOS” (FABENI, 2012) e “Jogo de Letras/Números Voltado para a Tecnologia Assistiva no Android ” (REETZ, 2013). Um protótipo de jogos 2D para a plataforma iOS voltado a educação assistiva. O projeto trabalha principalmente com pranchas, planos e símbolos. O jogo consiste em desenhar os números de 0 a 9 em um plano que pode ser montado pelo usuário e permite escolher o traçado, o predador e a presa. O projeto também permite trocar o plano de fundo da tela principal e possui um áudio que executa durante jogo. Mesmo que tenha ficado lento entre transições e consumindo muita memória, o jogo acabou se mostrando funcional. Este trabalho também é um complemento do framework Tagarela que é mantido pelo Grupo de Pesquisa em Computação Gráfica, Processamento de Imagens e Entretenimento Digital da FURB.

Palavras-chave: Jogos 2D. IOS. Tagarela.

ABSTRACT

This work is an extension of the " Aplicativo para comunicação alternativa no iOS " (Fabeni, 2012) and " Jogo de Letras/Números Voltado para a Tecnologia Assistiva no Android" (REETZ, 2013). A 2D game prototype for the iOS platform facing assistive education. The project works primarily with boards, plans and symbols. The game is to draw the numbers 0-9 on a plan that can be assembled by the user and allows you to choose the path, the predator and prey. The project also allows changing the main screen background and has an audio performing during the game. Even though it has been slow transitions between and consuming a lot of memory, the game turned out to be functional. This work is also a Jabber framework complement that is maintained by the Research Group on Computer Graphics, Image Processing and Entertainment Digital FURB.

Keywords: 2D games. IOS. Tagarela.

LISTA DE FIGURAS

Figura 1 – Símbolos que compõem o `plano` escolhido. 16
Figura 2 – Sessão de mais acessados no aplicativo 17
Figura 3 – Aplicativo Desenhe e Aprenda a Escrever 18
Figura 4 – Desenhando o `símbolo` 19
Figura 5 – Diagrama de casos de uso 21
Figura 6 – Diagrama de pacotes 25
Figura 7 – Classes do pacote Game 26
Figura 8 – Diagrama de sequências 27
Figura 9 – tela inicial do Tagarela 37
Figura 10 – Criando uma nova `prancha` 38
Figura 11 – Tela principal do jogo 40
Figura 12 – Interação com o `símbolo` principal 40
Figura 13 – `Plano` sendo concluído 41
Figura 14 – Níveis de processamento no iPad 42
Figura 15 – Monitoramento do nível de memória usada no iPad 43

LISTA DE QUADROS

Quadro 1 – Caso de uso Selecionar `plano` 21
Quadro 2 – Caso de uso Selecionar `prancha` 22
Quadro 3 – Caso de uso jogar `plano` 22
Quadro 4 – Caso de uso alterar plano de fundo 23
Quadro 5 – Caso de uso alterar predador 23
Quadro 6 – Caso de uso alterar presa 24
Quadro 7 – Caso de uso alterar traço 24
Quadro 8 – Parte do método `loadSelectedPlan` 29
Quadro 9 – Parte do método `loadGameParts` 30
Quadro 10 – Método `makeWayPoints` 31
Quadro 11 – Método `touchesMoved` 32
Quadro 12 – Método `isWallPixel` 32
Quadro 13 – Método `touchesEnded` 33
Quadro 14 – Método nextPlanAnimation 33
Quadro 15 – Método `nextPlan` 34
Quadro 16 – Inicialização da classe `TGPreviewView` 35
Quadro 17 – Método `refreshPlan` 35
Quadro 18 – Método `playSoundFromGroupPlan` 36
Quadro 19 – Método `addOnHistoric` 36
Quadro 20 – Níveis de processamento durante o jogo 42
Quadro 21 – Nível de memoria em diferentes quantidades de pranchas 44
Quadro 22 – Mensagem de erro 414 45

SUMÁRIO

1 INTRODUÇÃO 12
1.1 OBJETIVOS 12
1.2 ESTRUTURA 13
2 FUNDAMENTAÇÃO TEÓRICA. 14
2.1 TECNOLOGIA ASSISTIVA E JOGOS EDUCACIONAIS 14
2.2 TAGARELA 15
2.3 TRABALHOS CORRELATOS 16
2.3.1 Livox 17
2.3.2 Desenhe e Aprenda a Escrever 17
2.3.3 Jogo de Letras/Números Voltado para a Tecnologia Assistiva no Android 18
3 DESENVOLVIMENTO 20
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA 20
3.2 ESPECIFICAÇÃO 20
3.2.1 Diagrama de casos de uso 20
3.2.1.1 Selecionar `plano` 21
3.2.1.2 Selecionar `prancha` 21
3.2.1.3 Jogar `plano` 22
3.2.1.4 Alterar plano de fundo 22
3.2.1.5 Alterar predador 23
3.2.1.6 Alterar presa 23
3.2.1.7 Alterar traço 24
3.2.2 Diagrama de pacotes 24
3.2.3 Diagrama de sequência 27
3.3 IMPLEMENTAÇÃO 28
3.3.1 Técnicas e ferramentas utilizadas 28
3.3.2 Classe `TGBoardInteractorViewController` 28
3.3.3 Classe `TGPreviewView` 34
3.3.4 Classe `TGHistoricView` 36
3.3.5 Operacionalidade do jogo 37
3.3.5.1 Tela inicial 37
3.3.5.2 Criando uma `prancha` para jogo 38

3.3.5.3 Tela principal do jogo 39
3.4 RESULTADOS E DISCUSSÕES 41
3.4.1 Memória e desempenho 41
3.4.1.1 Processamento 41
3.4.1.2 Memória 42
3.4.2 Comparativo entre o trabalho desenvolvido e os trabalhos correlatos 44
4 CONCLUSÕES 46
4.1 EXTENSÕES 46
REFERÊNCIAS BIBLIOGRAFICAS 47

1 INTRODUÇÃO

A crescente matrícula de alunos com deficiência em escolas de ensino comum tem feito com que estudiosos e educadores discutam questões que envolvem ensino, a aprendizagem e a avaliação desses alunos (CORREIA, V., 2013). Entre as dificuldades que pessoas especiais comumente sofrem uma delas é a comunicação. E apesar da tecnologia existente, a fala continua sendo a principal forma de comunicação. Contudo existem pessoas que, devido a fatores neurológicos, físicos, emocionais e cognitivos, se mostram incapazes de se comunicar de maneira oral (LORENA, 2010).

Quando o quesito é comprometimento da fala em alunos tanto escolas como educadores apresentam uma fragilidade principalmente entre desconhecimento de formas de comunicação expressiva e receptiva dos alunos. O desconhecimento da existência e/ou uso de recursos de Tecnologia Assistiva (TA) como facilitadores de processos comunicativos conta como uma fragilidade também (CORREIA, V., 2013).

Atualmente, *tablets* e *smartphones* estão tendo um papel importante na área de TA. Os sistemas de Comunicação Alternativa e Amplificada (CAA) que antes eram restritos ao computador sofreram uma revolução com a chegada dos dispositivos móveis (CORREIA, P., 2013). O Instituto de Tecnologia Social (ITS BRASIL, 2008) afirma que neste cenário de criação de tecnologias que garantam a acessibilidade, a CAA tem contribuído para facilitar e efetivar a comunicação das pessoas com ausência ou prejuízo da fala. Existem mais de 300 (trezentos) aplicativos voltados a área de CAA a um custo consideravelmente baixo se for comparado aos tradicionais sistemas de comunicação (CORREIA, P., 2013).

Em 2013 o acadêmico Wagner Jean Reetz desenvolveu um jogo 2D para o framework Tagarela, feito para Android. O objetivo deste jogo, segundo Reetz (2013), foi desenvolver um protótipo voltado para a TA explorando o aspecto pedagógico/lúdico em computação aplicada onde o cenário é um jogo 2D que manipula letras e números. E no trabalho de Fabeni (2012), foi feito o projeto Tagarela (2014) para iOS. Diante do exposto, esse projeto foi aprimoramento do projeto de Fabeni, refazendo o projeto de Reetz de um jogo 2D para iOS.

1.1 OBJETIVOS

O objetivo desse trabalho é aprimorar o “Aplicativo para comunicação alternativa no iOS” (FABENI, 2012) refazendo o “Jogo de Letras/Números Voltado para a Tecnologia Assistiva no Android ” (REETZ, 2013) feito na plataforma Android para iOS.

Os objetivos específicos do trabalho são:

a) migrar o “Jogo de Letras/Números voltado para a TA no Android ” (REETZ,

2013) para iOS;

b) permitir que o usuário escolha o predador, presa, plano de fundo e traçado durante o jogo;
c) analisar a possibilidade de utilizar um sintetizador de voz para reproduzir o significado das palavras ou frases nos `planos` customizados.

1.2 ESTRUTURA

Este trabalho está dividido em quatro capítulos. O segundo capítulo aborda a fundamentação teórica necessária para compreensão deste trabalho como a tecnologia assistiva e jogos 2D e o aplicativo Tagarela. O terceiro é sobre o desenvolvimento do projeto como especificações, os diagramas para melhor entendimento e a implementação com alguns quadros com código fonte. No terceiro capítulo encontram-se também os resultados e discussões demonstrando o desempenho do aplicativo no iPad e no quarto capítulo, as conclusões e extensões para o projeto.

2 FUNDAMENTAÇÃO TEÓRICA.

A seção 2.1 trata sobre Tecnologia Assistiva e jogos educacionais, também expondo as ideias de Jogos 2D. A seção 2.2 descreve o framework Tagarela e ao final, a seção 2.3 apresenta alguns trabalhos correlatos.

2.1 TECNOLOGIA ASSISTIVA E JOGOS EDUCACIONAIS

Segundo Bersh e Pelosi (2011) Tecnologia Assistiva (TA) são recursos e serviços que tem como objetivo melhorar as habilidades funcionais de uma pessoa com deficiência ou incapacidade por motivo de envelhecimento, tentando assim trazê-lo para uma qualidade de vida melhor e promover sua inclusão social. Quem trabalha com TA tem responsabilidade de avaliar o paciente e prescrever o recurso apropriado. A equipe de TA envolve vários profissionais de diferentes áreas como professores, terapeutas ocupacionais, fonoaudiólogos, fisioterapeutas, engenheiros, entre outras áreas (BERSH, PELOSI 2011). Neste cenário de criação de tecnologias a CAA tem contribuído para facilitar a comunicação das pessoas com ausência ou prejuízo na fala (ITS BRASIL, 2008, p.11).

A Comunicação Alternativa (CA) são gestos manuais, expressões faciais, corporais, gravuras, figuras gráficas, objetos reais, entre outros para comunicação presencial entre indivíduos incapazes de usar a linguagem oral (ITS BRASIL, 2008, p.11). Lorena (2010) explica também que é considerada Comunicação Ampliada quando o indivíduo não possui comunicação suficiente através da fala e/ou escrita e CA quando o indivíduo não apresenta outra forma de comunicação.

Com o avanço da tecnologia, vários recursos começaram a ser explorados na área de CA e educação, como os jogos. Segundo Gebran (2009), os jogos educacionais se tornam ferramentas ideais para a aprendizagem, com o objetivo de estimular o interesse do aluno e auxiliando na construção de novas descobertas. Cuperschmid e Hildebrand (2013, p. 36) apontam que os jogos educacionais em meio eletrônico possuem toda a característica e ambiente de um jogo, mas com o objetivo de ensinar. Combinando aprendizado e entretenimento, mesmo em assuntos desmotivadores, o envolvimento, a interatividade e a participação ativa comprovam a eficiência do aprendizado (CUPERSCHMID; HILDEBRAND, 2013, p. 37).

Entre as opções de jogos educacionais existem os jogos bidimensionais. Para os jogos 2D usam-se basicamente imagens. Tanto para personagens, quanto para plano de fundo e até mesmo em objetos de interação usando principalmente *sprites* (FEIJÓ; SILVA; CLUA, 2009, p. 127). *Sprites,* segundo Estrella at al. (2010 p. 50), são imagens que podem ser manipuladas

independentes do restante do jogo. Ainda se fala de *sprites* animados que são *sprites* que mudam de forma a cada determinado intervalo de tempo (ESTRELLA at al., 2010, p 50). “Um jogo 2D basicamente consiste em manipular, mover, mostrar, esconder, bater, matar *sprites*...” (FEIJÓ; SILVA; CLUA, 2009, p. 128). Os autores ainda explicam que o conceito de um *sprite* pode ser comparado ao de uma classe: um jogo pode ter vários objetos gráficos, cada um sendo uma imagem, com seus atributos como, tamanho, rotação e posição, que podem ser atualizados constantemente. Outro importante conceito é a imagem de fundo, que podem ser estáticas ou se mover, dando a impressão de que é a câmera que está se movendo (FEIJÓ; SILVA; CLUA, 2009, p. 128).

Os *sprites* não precisam ter um formato retangular como costumam ter as imagens. Para isso alguns formatos de imagens usam um elemento a mais para cada *pixel*. Além do RGB, cores fundamentais do pixel: vermelho (Red), verde (Green) e azul (Blue), esses padrões permitem o elemento alpha também. Esse quarto valor indica o quanto o *pixel* é transparente, tornando-o não visível (FEIJÓ; SILVA; CLUA, 2009, p. 145).

Assim, de uma forma geral, os jogos educacionais podem ser ferramentas eficientes, pois eles divertem enquanto motivam, facilitam o aprendizado e podem aumentar a capacidade de retenção do que foi ensinado (TAROUCO at al., 2004). Um exemplo de jogo educacional é o *framework* Tagarela que será descrito a seguir (REETZ, 2013).

2.2 TAGARELA

Segundo Fabeni (2012) o Tagarela é um aplicativo que fornece uma plataforma para que as pessoas envolvidas no dia a dia do paciente consigam comunicar-se entre si de forma que haja uma evolução na capacidade de comunicação do paciente, utilizando `planos` de atividades elaborados pelo fonoaudiólogo em conjunto com o tutor do paciente.

O aplicativo foi desenvolvido para a plataforma iOS, mas está em desenvolvimento para Android e web. O aplicativo tem como características principais: criar `símbolos` personalizados, associar áudio aos `símbolos`, troca de mensagens entre `usuários`, criar `plano` de atividades, entre outras (REETZ, 2013). Na Figura 1 pode ser visto como funciona o aplicativo, mostrando os `símbolos` que compõe uma `prancha` de um determinado `plano`. Nesta figura também pode ser observado o uso de cores nas bordas das imagens para determinar os grupos que as mesmas pertencem (sujeito, verbo, etc.).

Um dos diferenciais do aplicativo são as funcionalidades de histórico de observações do paciente e o compartilhamento de recursos como `planos`, `pranchas` e `símbolos` (FABENI, 2012).

Os símbolos são uma representação visual de um conceito ou um objeto da vida real. Esse símbolo também é associado a um áudio, induzindo o usuário a assimilar o som a imagem. As pranchas são os agrupamentos de símbolos que possibilitam a criação de uma sentença visual que ajuda na evolução da comunicação do usuário. Os planos são os agrupamentos de pranchas que possuem as mesmas finalidades das pranchas em um nível mais elevado de comunicação (TAGARELA, 2013).

Com o uso do aplicativo em sessões de terapia, foi comprovado que o aplicativo pode realmente trazer benefícios para as pessoas envolvidas no processo de comunicação do paciente, promovendo uma maior inclusão social do mesmo (FABENI, 2012).

Figura 1 – Símbolos que compõem o plano escolhido.

Fonte: Fabeni (2012, p.79).

O trabalho de Fabeni (2012) permitiu iniciar a criação do *framework*, que tem como principal função disponibilizar uma Rede Social de Comunicação Alternativa e Humanizada para auxiliar crianças portadoras de necessidades especiais (TAGARELA, 2013).

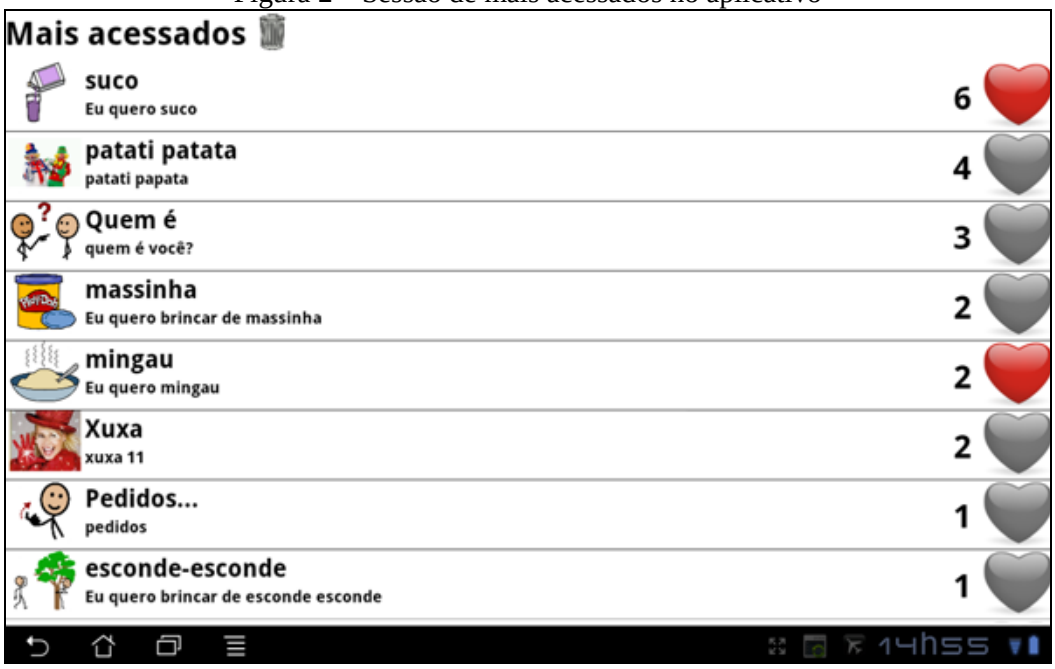
2.3 TRABALHOS CORRELATOS

Existem alguns aplicativos desenvolvidos para Comunicação Alternativa. Alguns deles são Livox, Desenhe e Aprenda a Escrever e a expansão do aplicativo Tagarela. Eles serão apresentados nessa seção.

2.3.1 Livox

Desenvolvido pela Agora Eu Consigo Tecnologias de Inclusão Social Ltda, o Livox é um aplicativo comercial que permite edição de pranchas, que são os conjuntos de símbolos. Possui também um modo de escrita livre com um sintetizador de voz que ajuda pessoas que por algum motivo perderam a voz (LIVOX, 2014). Este aplicativo conta com várias pranchas prontas e imagens para adicionar em novas pranchas, mas podem ser adicionadas fotos também (LIVOX, 2014). O aplicativo pode ser visto na Figura 2.

Figura 2 – Sessão de mais acessados no aplicativo



Fonte: Livox (2014)

Apesar de ser desenvolvido para Android o aplicativo não está disponível na Play Store. Ele é vendido junto com um treinamento que é dado ao paciente. O aplicativo possui uma sessão de mais acessados, como visto na Figura 2. Ele armazena o número de vezes que a prancha foi selecionada, podendo adicioná-la as favoritas, ficando mais acessível (LIVOX, 2014).

2.3.2 Desenhe e Aprenda a Escrever

O Desenhe e Aprenda a Escrever é um jogo desenvolvido pela FizzBrain para a plataforma iOS (FIZZBRAIN, 2013). Foi desenvolvido por dois professores americanos que possuem quase 50 anos de experiência em educação infantil, usando suas técnicas educacionais avançadas nas escolas americanas, ensinando as crianças a escrever (FIZZBRAIN, 2013). O aplicativo pode ser visto na Figura 3 exibindo o menu principal e uma parte do ambiente de jogo no qual o usuário escreve um texto customizado

Figura 3 – Aplicativo Desenhe e Aprenda a Escrever

Fonte: FizzBrain (2013).

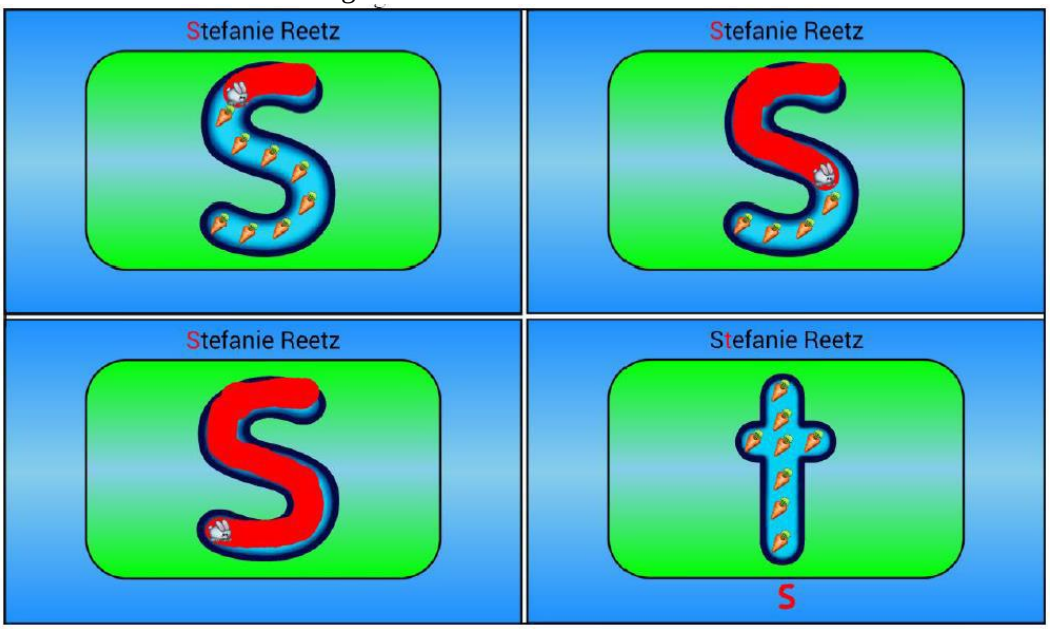
O jogo procura através do entretenimento, auxiliar o aprendizado da criança em um ambiente interativo. O jogo possui além da lista de alfabeto e de números, uma lista de nomes de animais e ainda permite adicionar palavras customizadas.

2.3.3 Jogo de Letras/Números Voltado para a Tecnologia Assistiva no Android

É uma expansão do aplicativo Tagarela, que segundo Reetz (2013) é um jogo 2D que permite desenhar letras e números. O aplicativo possui listas de `planos` personalizáveis permitindo escrever nomes ou apenas seguir o alfabeto ou os números. Também possui a funcionalidade de gravar os textos customizados e escrever futuramente.

O jogo utiliza alguns cenários e atividades que permitem trabalhar com todas as letras e números (REETZ, 2013). O jogo trabalha com agrupamento de `pranchas`, ou `planos`. Cada `prancha` possui uma letra ou um número que pode ser editado pelo `usuário`. A Figura 4 exemplifica o desenho de um `símbolo`.

Figura 4 – Desenhando o símbolo



Fonte: Reetz (2013).

No momento que o usuário escolhe um conjunto de planos o conjunto é carregado na parte superior da tela e o primeiro plano é carregado ao centro. Com base na propriedade alpha dos *pixels*, são desenhados os pontos de interesse do símbolo mostrados na figura 4 como cenouras dentro da letra (REETZ, 2013). Cada toque efetuado na tela pelo usuário é capturado, validado e armazenado numa lista de pontos. Então a lista de pontos será lida e desenhada na tela, mostrando o caminho que o usuário passou pelo símbolo (REETZ, 2013). Quando o usuário passar por todos os pontos de interesse, desenhando a letra ou número, o plano é concluído. Nesse momento, o aplicativo faz a leitura do áudio do símbolo e vai para o próximo plano (REETZ, 2013). Ao final do plano, o usuário pode ver o que ele escreveu na parte inferior da tela e ao concluir todas, mostra o que o usuário escreveu no centro da tela (REETZ, 2013).

3 DESENVOLVIMENTO

Nesse capítulo serão mostradas as especificações do problema na seção 3.1. A seção 3.2 exibe a especificação do projeto com diagramas para melhor entendimento. A seção 3.3 explica partes da implementação e por fim, a seção 3.4 relata os resultados e discussões.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA

O jogo 2D tem como principais funções:

a) utilizar vários cenários de atividades que permitam trabalhar com símbolos (Requisito Funcional - RF);
b) permitir a escolha do símbolo de fundo durante o jogo (RF);
c) ter melodia de fundo definido pelo símbolo de fundo (RF);
d) permitir a escolha de um `símbolo` definir o traçado para o caminho do `símbolo` durante o jogo (RF);
e) permitir a escolha de um `símbolo` para definir o `predador` durante o jogo (RF);
f) permitir a escolha de um `símbolo` para definir a `presa` durante o jogo (RF);
g) ser implementado utilizando o ambiente Xcode 6.0 (Requisito Não Funcional – RNF);
h) utilizar a linguagem Objective-c (RNF);
i) permitir executar no sistema operacional iOS 7 ou superior (RNF);
j) manter a sincronização dos `planos` com o servidor web (RNF).

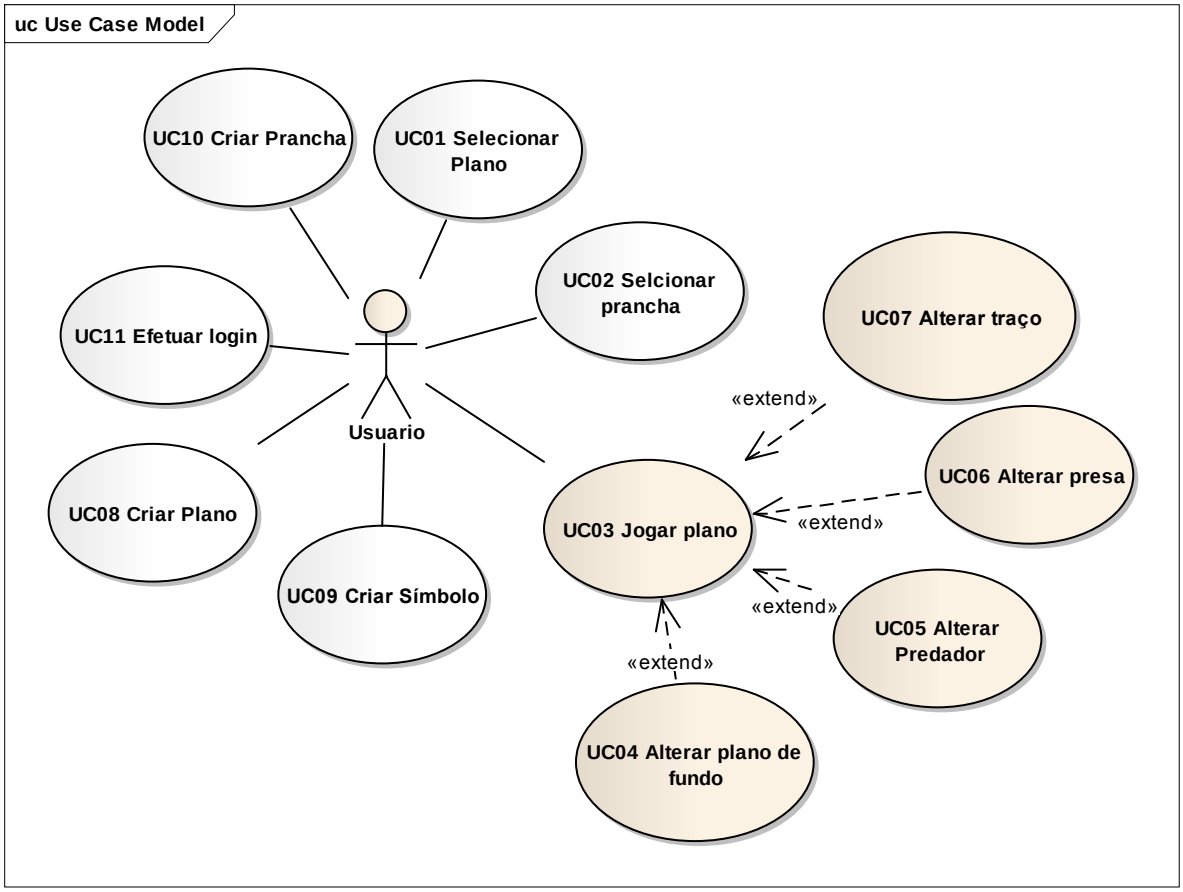
3.2 ESPECIFICAÇÃO

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

Nessa seção são descritas todas as funcionalidades do jogo. Apenas um ator foi identificado para o uso do jogo. O ator `usuário` pode ser visto na Figura 5 com os casos de uso sendo que os casos de uso `Selecionar plano`, `Selecionar prancha`, `Criar símbolo`, `Criar plano`, `Criar prancha` e `Efetuar login` foram reutilizados.

Figura 5 – 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 (FABENI, 2013) e possuir base de dados sincronizada.
Cenário Principal	1. A aplicação Tagarela (FABENI, 2013) apresenta os `planos` disponíveis para o `Usuário`. 2. O `Usuário` seleciona o `plano` desejado.
Pós-condições	A lista de `pranchas` do `plano` é exibida.

3.2.1.2 `Selecionar prancha`

Este caso de uso descreve a relação entre o `Usuário` e a funcionalidade que possibilita selecionar uma `prancha`. O Quadro 2 descreve em detalhes este caso de uso.

Quadro 2 – Caso de uso Selecionar prancha

UC02 – Selecionar prancha	
Pré-condições	1. Usuário selecionou um plano da lista do tipo jogo.
Cenário Principal	1. A aplicação Tagarela (FABENI, 2013) apresenta a lista de pranchas do plano selecionado. 2. O Usuário seleciona a prancha desejada.
Cenário alternativo	O usuário pode tocar em voltar e retorna para o UC01.
Pós-condições	A aplicação exibe a tela inicial do jogo com a prancha selecionada em destaque.

3.2.1.3 Jogar plano

O caso de uso jogar plano descreve a principal interação do Usuário com o aplicativo. O Quadro 3 ilustra esse caso de uso

Quadro 3 – Caso de uso jogar plano

UC03 – Jogar plano	
Requisito atendido	RF01
Pré-condições	O Usuário deve ter feito o UC01 e UC02
Cenário Principal	1.O jogo exibe na parte superior da tela a pré-visualização de todos os símbolos contidos nas pranchas do plano. 2.O jogo inicia o áudio do plano de fundo. 3.O jogo mantém destacado em vermelho o símbolo atual na pré-visualização de pranchas contida 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 todos os pontos existentes dentro do símbolo. 7.O jogo reproduz o áudio do símbolo desenhado pelo usuário. 8.O jogo aplica efeito de transformação no símbolo que diminui de tamanho e desce até a área inferior da tela. 9.O jogo exibe símbolo escrito pelo usuário na parte inferior da tela. 10.Se ainda houver pranchas para serem jogadas, o jogo destaca a próxima prancha a ser jogada e volta ao passo 2 do cenário principal. 11.O usuário concluí todos as pranchas contidas no plano.
Fluxo alternativo 1	O usuário pode optar por não fazer todas as pranchas. Ao tocar em voltar ele retorna para o menu principal.
Fluxo alternativo 2	O usuário pode passar para a próxima prancha sem fazer a atual tocando no botão para a próxima prancha.
Fluxo alternativo 3	O usuário pode voltar para a prancha anterior tocando no botão para a prancha anterior.
Pós-condições	O jogo exibe o histórico no centro da tela e toca o áudio do plano

3.2.1.4 Alterar plano de fundo

O usuário pode em algum momento do jogo alterar o plano de fundo da tela principal. O Quadro 4 explica de forma detalhada a sequência.

Quadro 4 – Caso de uso `alterar plano de fundo`

UC04 – `Alterar plano de fundo`	
Requisito atendido	RF02, RF03
Pré-condições	O `usuário` deve estar no cenário principal do UC03
Cenário Principal	1.O `usuário` toca no ícone de alterar plano de fundo. 2.Uma janela de seleção de `categorias` aparece no meio da tela. 3.O `usuário` seleciona a `categoria` do `símbolo`. 4.O jogo atualiza a tela exibindo os `símbolos` da `categoria` selecionada. 5.O `usuário` seleciona o `símbolo` desejado.
Fluxo alternativo	A qualquer momento do cenário principal o `usuário` pode cancelar a ação tocando em voltar ou fora da tela de seleção.
Pós condições	1.O jogo altera o `símbolo` de plano de fundo. 2.O jogo passa a reproduzir o áudio do `símbolo` selecionado.

3.2.1.5 `Alterar predador`

O `usuário` pode em algum momento do jogo alterar o `predador` que representa o `usuário` dentro do jogo. O Quadro 5 explica de forma detalhada a sequência.

Quadro 5 – Caso de uso `alterar predador`

UC05 – `Alterar predador`	
Requisito atendido	RF05
Pré-condições	O usuário deve estar no cenário principal do UC03
Cenário Principal	1.O `usuário` toca no ícone de predador. 2.Uma janela de seleção de `categorias` aparece no meio da tela. 3.O usuário seleciona a `categoria` do `símbolo`. 4.O jogo atualiza a tela exibindo os `símbolos` da `categoria` selecionada. 5.O usuário seleciona o `símbolo` desejado.
Fluxo alternativo	A qualquer momento do cenário principal o `usuário` pode cancelar a ação tocando em voltar ou fora da tela de seleção.
Pós condições	O jogo altera o `símbolo` que define o predador

3.2.1.6 Alterar presa

O `usuário` tem a possibilidade de alterar a `presa` durante o jogo. O Quadro 6 explica de forma detalhada a sequência.

Quadro 6 – Caso de uso `alterar presa`

UC05 – `Alterar presa`	
Requisito atendido	RF06
Pré-condições	O usuário deve estar no cenário principal do UC03
Cenário Principal	1.O `usuário` toca no ícone de presa. 2.Uma janela de seleção de `categorias` aparece no meio da tela. 3.O `usuário` seleciona a `categoria` do `símbolo`. 4.O jogo atualiza a tela exibindo os `símbolos` da `categoria` selecionada. 5.O usuário seleciona o `símbolo` desejado.
Fluxo alternativo	A qualquer momento do cenário principal o `usuário` pode cancelar a ação tocando em voltar ou fora da tela de seleção.
Pós condições	O jogo altera o `símbolo` que define a presa.

3.2.1.7 `Alterar traço`

Durante o cenário principal do UC03 o usuário pode alterar o `traçado` que é exibido na tela. O Quadro 7 explica de forma detalhada a sequência.

Quadro 7 – Caso de uso `alterar traço`

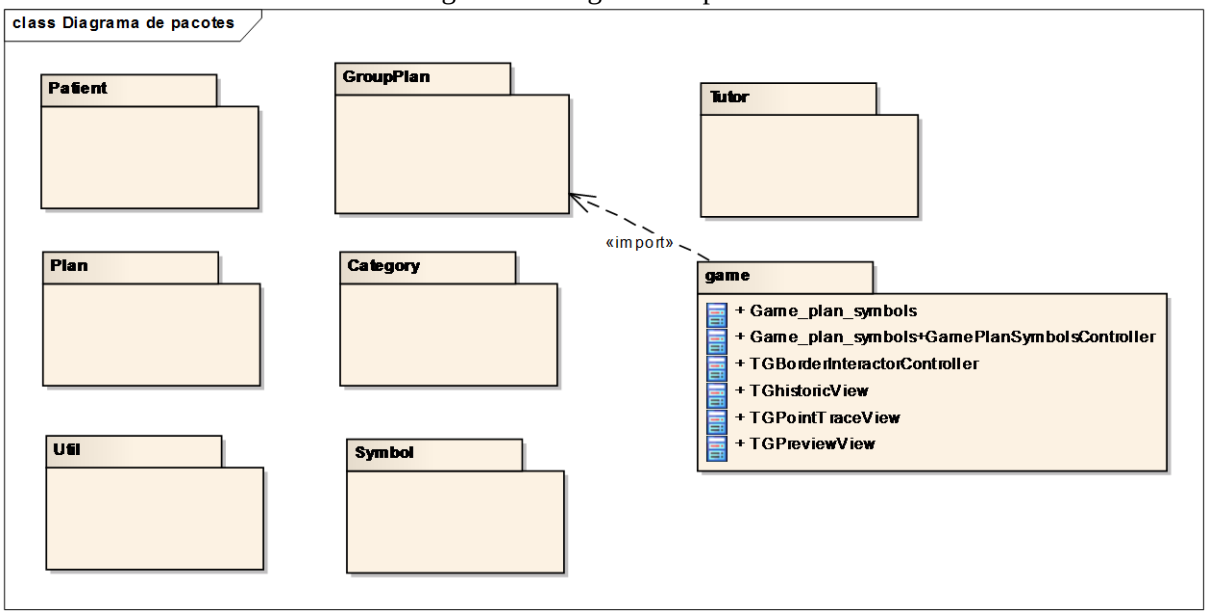
UC07 – `Alterar traço`	
Requisito atendido	RF04
Pré-condições	O `usuário` deve estar no cenário principal do UC03
Cenário Principal	1.O `usuário` toca no ícone de alterar traço. 2.Uma janela de seleção de `categorias` aparece no meio da tela. 3.O `usuário` seleciona a `categoria` do `símbolo`. 4.O jogo atualiza a tela exibindo os `símbolos` da `categoria` selecionada. 5.O `usuário` seleciona o `símbolo` desejado.
Fluxo alternativo	A qualquer momento do cenário principal o `usuário` pode cancelar a ação tocando em voltar ou fora da tela de seleção.
Pós condições	1.O jogo altera o `símbolo` do traçado. 2.O jogo altera o áudio do traçado para o som do `símbolo` selecionado.

Os casos de uso `criar símbolo`, `criar plano`, `criar prancha` e `efetuar login` já estavam presentes no projeto Tagarela (2014). Para mais informações consultar o trabalho de Fabeni (2012).

3.2.2 Diagrama de pacotes

O diagrama de pacotes apresenta uma visão de como as classes estão agrupadas. Apenas o pacote `Game` faz parte desse trabalho, mas outros pacotes do projeto Tagarela foram utilizados. A Figura 6 mostra o diagrama de pacotes baseado no diagrama de Fabeni (2012). Nela pode ser vista a nova associação feita com o pacote `GroupPlan`.

Figura 6 – Diagrama de pacotes



O pacote `Patient` é responsável por criar, persistir e exibir os dados sobre os pacientes. Esses dados são criados pelo fonoaudiólogo responsável pelo paciente (FABENI, 2012).

O pacote `Category` tem como funcionalidade exibir as categorias de `símbolos` criadas pelo fonoaudiólogo. As classes do pacote também são responsáveis por criar e manter os dados (FABENI, 2012).

As classes do pacote `Symbol` são responsáveis por criar, exibir e manter os dados dos `símbolos`. Eles são criados pelo fonoaudiólogo também (FABENI, 2012).

O pacote `Tutor` tem como função lidar com as informações relativas ao tutor criado pelo fonoaudiólogo. As classes desse pacote têm como função manter, criar e exibir as informações referentes aos tutores (FABENI, 2012).

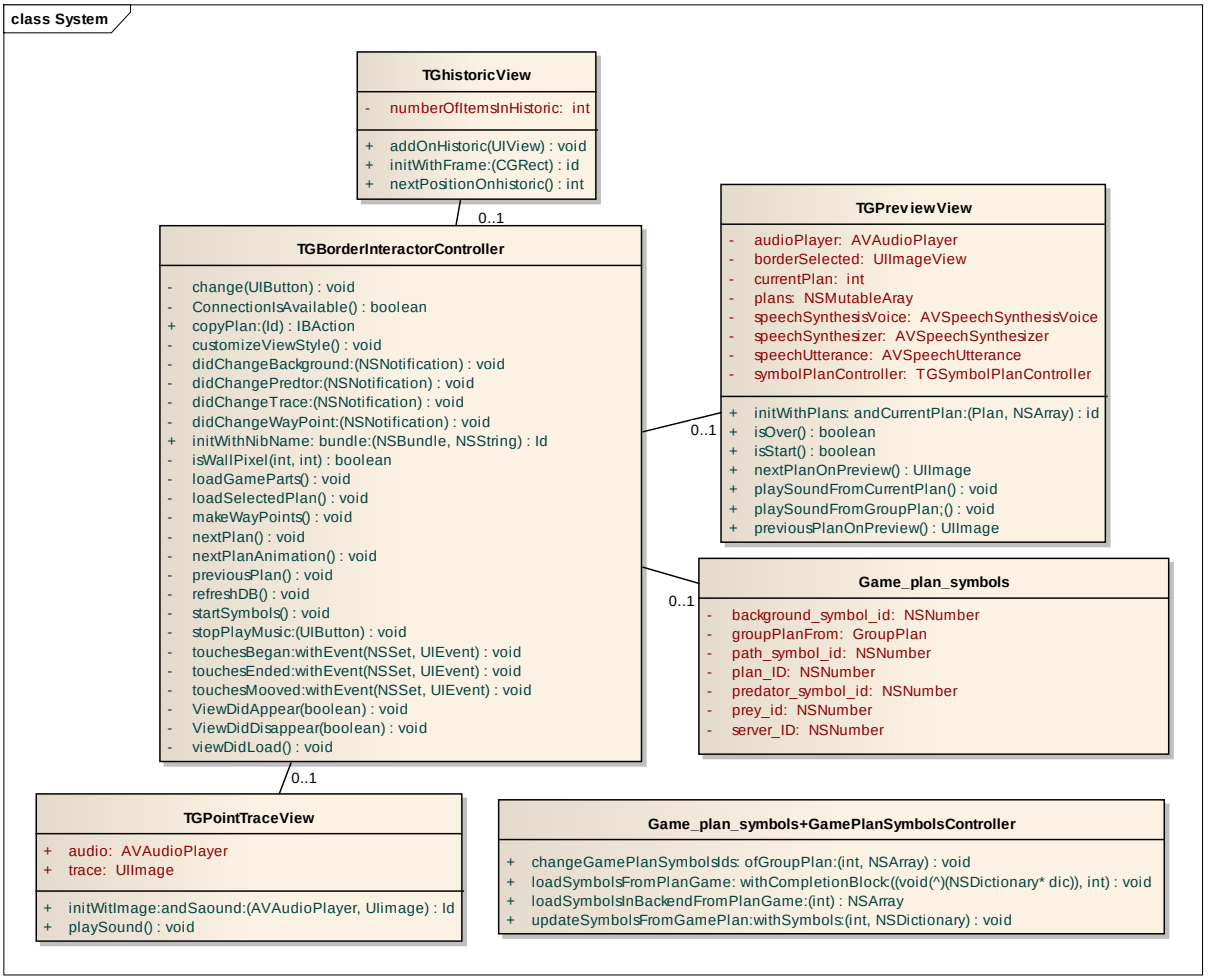
O pacote `Plan` tem como função lidar com as informações dos conjuntos de `símbolos` ou `pranchas` do `usuário`. Essas classes têm como função criar, manter e mostrar as planchas criadas pelo fonoaudiólogo (FABENI, 2012).

O pacote `GroupPlan` lida com as informações dos conjuntos de `pranchas` ou `planos` criados pelo fonoaudiólogo. Essas classes têm como função exibir, manter e criar novos `planos`.

O pacote `Util` são classes que fornecem utilidade para o aplicativo. Ele trabalha principalmente com reestruturação de imagens e transições de objetos `NSData` para *String* assim como o oposto.

A Figura 7 mostra as classes do pacote que guarda as classes principais do jogo. A classe `TGBorderInteractorController` teve os seus atributos ocultados para melhor exibição.

Figura 7 – Classes do pacote Game



A classe `TGPreviewView` herda da classe `UIScrollView` e tem como função principal fazer todo o gerenciamento das `pranchas` existentes no `plano`. Essa classe é exibida na parte superior da tela mostrando todas as `pranchas` do `plano`. Ela mostra para o `usuário` qual é a `prancha` que ele está. Ela também é responsável por fornecer a próxima `prancha` ou a anterior dependendo da vontade do `usuário`. É de responsabilidade dessa classe também informar se é o final do jogo ou não. Outra função da classe é tocar o áudio do `símbolo` da `prancha` e ao final do `plano` o áudio da junção de todas as `pranchas`.

A classe `TGHistoricView` herda de `UIScrollView` e exibe todo o desenho feito ao final da `prancha`. Cada `prancha` finalizada é exibida na parte inferior da tela. O `símbolo` da `prancha` não é exibido nesse momento, apenas o traçado que o `usuário` fez. É importante ressaltar que apenas as `pranchas` finalizadas, ou seja, com todas as `presas` capturadas, são enviadas para essa classe.

A classe `TGGamePointTraceView` também herda de `UIView` e é responsável por armazenar a imagem e áudio do `símbolo` responsável pelo traçado.

A classe `Game_plan_symbols+GamePlanSymbolsController` é responsável por se comunicar com o banco dados. Ele faz a persistência e a consulta ao banco para os `símbolos` de `plano de fundo`, `predador`, `presa` e `traçado`.

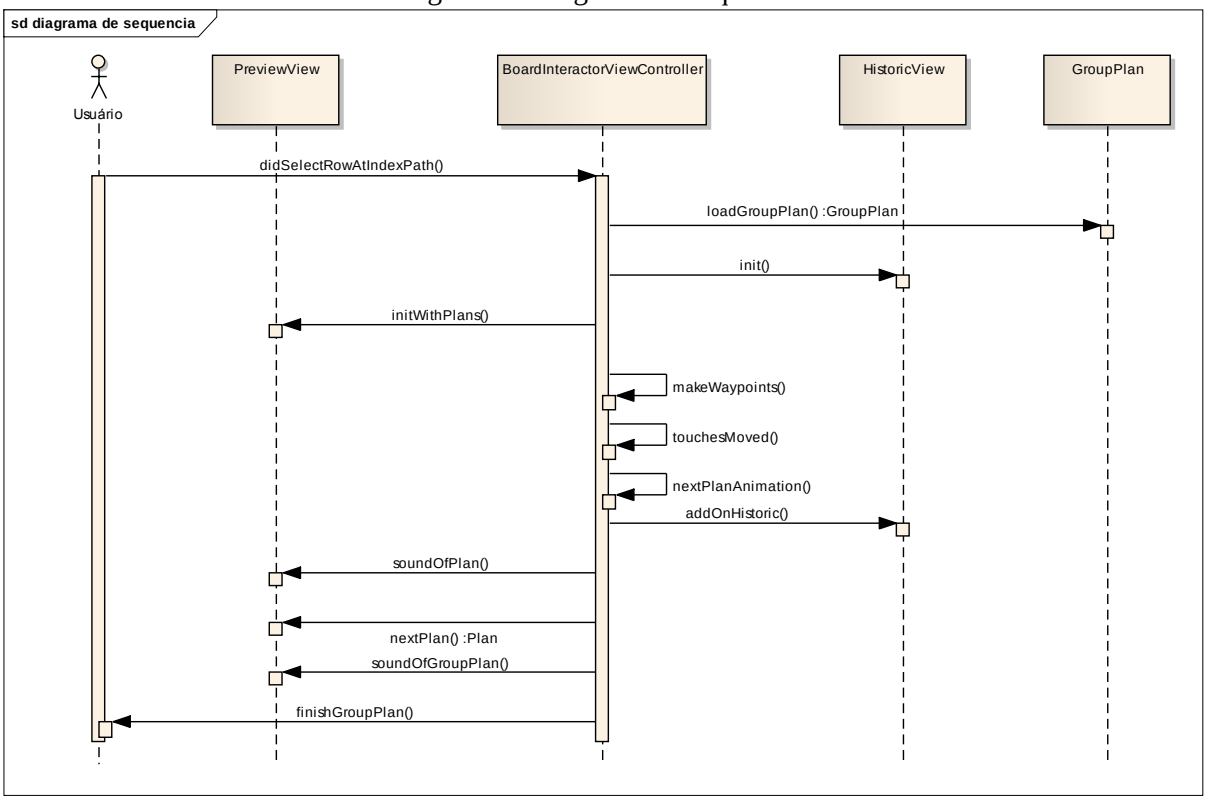
A classe `Game_plan_symbols` é uma classe gerada pelo Xcode quando uma nova entidade é criada no Core Data. Essa classe possui apenas os métodos e atributos que dão acesso ao objeto.

A classe `TGBoardInteractorController` é uma classe que herda de `UIViewController` e já existia no projeto de Fabeni (2012). Ela é responsável por fazer a interação das pranchas com o usuário, verificar e carregar o respectivo tipo do `plano`. É nessa classe que é feito a verificação do `alpha` e recebe o desenho feito pelo `usuário`.

3.2.3 Diagrama de sequência

O diagrama de sequência demonstra qual a interação do *Usuário* com o jogo relatado neste trabalho. A ilustração do diagrama de sequência pode ser vista na Figura 8.

Figura 8 – Diagrama de sequência



Quando o usuário seleciona uma `prancha` que possui a parte de jogo o aplicativo começa a carregar as partes exclusivas do jogo. O método `LoadGroupPlan():groupPlan`

mostrado na Figura 8 é chamado trazendo todas as pranchas do plano do banco de dados. A parte do historicView é iniciada e os planos são passados para a PreviewView no método de inicialização chamado initWithPlans. O método makeWayPoints é chamado para adicionar os pontos de interesse que serão as presas. Depois de concluído e as presas posicionadas o usuário pode começar a sua interação com o símbolo. Cada vez que o usuário faz um toque de movimento na tela a classe BoardInteractorViewController detecta e chama o método touchesMoved. Esse insere o traçado e verifica se ainda existem presas para serem coletadas. Se não existirem mais o método nextPlanAnimation faz o efeito de passagem do desenho feito pelo usuário para a classe HistoricView que recebe uma UIView através do método addOnHistoric. O método soundOfPlan é chamado para tocar o áudio do símbolo e o nextPlan():plan é chamado para enviar a nova prancha que estará no centro da tela. Ao final do plano a classe PreviewView usa o sintetizador de voz para falar o nome de todas as pranchas no plano e o plano é finalizado.

3.3 IMPLEMENTAÇÃO

A seguir serão mostradas as classes e partes de código que foram implementadas no jogo 2D.

3.3.1 Técnicas e ferramentas utilizadas

A linguagem utilizada para o desenvolvimento do jogo 2D foi o Objective C. O projeto Tagarela é feito com executa no iOS 8.1 que era no qual ele já estava sendo executado e utilizando o Xcode 6.0 como ambiente de desenvolvimento. O simulador iOS foi utilizado para depuração e o dispositivo utilizado para testes foi o Ipad 2.

3.3.2 Classe TGBoardInteractorViewController

A TGBoardInteractorViewController é uma classe originária do framework Tagarela de Fabeni (2012). Ela é responsável por fazer toda a interação com o usuário. Nela foram implementados os métodos principais do jogo 2D.

O método loadSelectedPlan foi implementado pelo acadêmico Fabeni (2012), mas teve uma pequena modificação para suportar o jogo 2D. No Quadro 8 é exibida a parte modificada que verifica se a variável type é a mesma que identifica como jogo e então o método loadGameParts é chamado. Se o plano não é do tipo jogo será exibida apenas a prancha com os seus símbolos.

Quadro 8 – Parte do método `loadSelectedPlan`

```
83      if ([_groupPlanController groupPlanForPlanWithPlanID:[[self selectedPlan]
        serverID]].type==1) {
85                  self.isGame = YES;
86              }
```

O método `loadGameParts` é responsável por exibir as novas modificações na tela. Ele é chamado apenas se o `plano` atual é do tipo jogo. Caso contrário é exibido apenas a `prancha` atual. O método tem a função de exibir os botões de controle do jogo que são: as setas para a `prancha` anterior e a próxima e os quatro botões para escolher os `símbolos` de `plano de fundo`, `predador`, `presa` e o `traçado` que são carregados do banco de dados se existirem. Ainda nesse método os objetos das classes `TGPreviewView`, `TGHistoricView` e `TGPointTraceView` são iniciados para a exibir a `pré-visualização` na parte superior, o histórico na parte inferior e o traçado que o `usuário` fará.

O Quadro 9 mostra a inicialização dos objetos e também o carregamento dos quatro `símbolos` que são trazidos do banco de dados. Esses `símbolos` podem ser vistos entre as linhas 508 e 511 e trazem respectivamente, a `imagem e áudio de fundo`, do `predador`, da `presa` e do `traço`. O áudio de erro, que é executado quando o usuário tentar fazer o traço fora do `símbolo`, também é carregado nesse momento.

Quadro 9 – Parte do método `loadGameParts`

```
494  self.drawView = [[UIView alloc]initWithFrame:imageView1.frame];
495      self.drawView.layer.zPosition = 2;
496      [self.view addSubview:self.drawView];
497
498      self.wayPoints = [[NSMutableArray alloc] init];
499
500      self.historicView=[[TGHistoricView alloc]initWithFrame:CGRectMake( 170,
     624, 700,100)];
501      [self.view addSubview:_historicView];
502
503      NSArray* arrayGroupPlans = [_groupPlanController
     loadPlansForGroupPlan:[_groupPlanController
     groupPlanForPlanWithPlanID:[[self selectedPlan] serverID]]
     andForSpecificTutor:[[[TGCurrentUserManager
     sharedCurrentUserManager]selectedTutorPatient]patientTutorID]];
504
505      self.previewView = [[TGPreviewView alloc]initWithPlans:arrayGroupPlans
     andCurrentPlan: self.selectedPlan];

506      [GamePlanSymbols loadSymbolsFromPlanGame:[[_groupPlanController
     groupPlanForPlanWithPlanID:[[self selectedPlan] serverID]] serverID]
     withCompletionBlobk:^(NSDictionary *symbolsGame) {
507          if (symbolsGame && self.isViewLoaded && self.view.window) {
508              _backgroundSymbol = [symbolPlanController
     loadSymbolsGameForGroupPlanId:(int)[[symbolsGame
     objectForKey:@"plan_background_symbol_id"] integerValue]];
509              _predatorSymbol = [symbolPlanController
     loadSymbolsGameForGroupPlanId:(int)[[symbolsGame
     objectForKey:@"predator_symbol_id"] integerValue]];
510              _wayPointSymbol =[symbolPlanController
     loadSymbolsGameForGroupPlanId:(int)[[symbolsGame
     objectForKey:@"prey_symbol_id"] integerValue]];
511              _traceSymbol = [symbolPlanController
     loadSymbolsGameForGroupPlanId:(int)[[symbolsGame
     objectForKey:@"path_symbol_id"] integerValue]];
512
513              [self startSymbols];
514                  }
515      }];
516      [self.view addSubview:_previewView];
```

No Quadro 9 também pode ser visto o carregamento das `pranchas` do `plano`. A linha 503 exibe um vetor que recebe todas as `pranchas` do banco de dados. Todos os métodos chamados nessa linha estavam presentes no projeto de Fabeni (2012).

O método `makeWayPoints` é um método baseado no algoritmo de Reetz (2013) para posicionar as presas nos locais corretos da imagem. Ele varre a imagem e em cada local que o valor do alpha é diferente de 0 ou 255 é criado uma `UIImageView` com a posição atual e adicionado ao `wayPoints`, o vetor das `presas`. Ao final elas são adicionadas a `view` que fica posicionada exatamente em cima da imagem do `símbolo`. O Quadro 10 mostra como o método localiza o ponto e adiciona no objeto `drawView`.

Quadro 10 – Método `makeWayPoints`

```
653   -(void)makeWayPoints{
654     [_wayPoints removeAllObjects];
655     for (int x = 0; x< imageView1.image.size.width; x++) {
656         for (int y = 0; y< imageView1.image.size.height; y++){
657           const UInt8* data = CFDataGetBytePtr(_pixelData);
658           int pixelInfo = ((imageView1.image.size.width  * y) + x) * 4;
659           int alpha = data[pixelInfo + 3];
660             if (alpha!=0 && alpha!=255) {
661                 //NSLog(@"%i , %i", x,y);
                    UIImageView *point2 = [[UIImageView
alloc]initWithImage:_wayPointImageView.image];
662                 point2.frame =CGRectMake(x/_scale-20, y/_scale-20, 40, 40);
                    [point2 setContentMode:UIViewContentModeScaleAspectFit];
663                 point2.clipsToBounds = YES;
                    [_wayPoints addObject:point2];
                }
664         }
665     }
666     for ( UIImageView *point in self.wayPoints) {
667         [self.drawView addSubview:point];
668     }
669     [self.drawView addSubview:_predatorView];
670 }
```

O método percorre todos os *pixels* da imagem para verificar o `alpha`. Quando encontra o `alpha` diferente de 0 ou 255 ele insere a imagem da presa que está na variável `wayPointImageView`. E na linha 633 a presa é adicionada ao `waypoints`. A última linha de código do quadro insere a imagem do `predador` na tela pois ela é sempre removida com o desenho feito da `prancha` anterior. Essa imagem só irá aparecer quando o `usuário` tocar e mover o dedo na tela.

O método `touchesMoved` é um método existente na classe `UIView`. Ele foi apenas sobrescrita para capturar a função de traço do `usuário`. Primeiramente, cada vez que o método é chamado um novo ponto é criado. Se o ponto for um caminho válido, ou seja, está dentro da imagem o predador é movido para esse ponto e uma `UIImageView` com a imagem do traço também é criada na mesma referência. Nesse momento também é verificado se esse ponto está em cima de alguma presa. Se estiver a `presa` é removida. Se não existir mais nenhuma `presa` o método `nextPlanAnimation` é chamado para trazer a nova `prancha`. Caso não seja o caminho certo apenas é executado o áudio do caminho errado. O Quadro 11 mostra parte do código do método.

Quadro 11 – Método `touchesMoved`

```
583 -(void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event{
584     if (self.isGame) {
585     CGPoint location = [[touches anyObject] locationInView:self.view];
586     CGRect fingerRect = CGRectMake(location.x-30 , location.y-30, 60, 60);
587 
588     if(CGRectIntersectsRect(fingerRect, _drawView.frame) && [self
    isWallPixel:location.x-258 :location.y-120]){
589         _predatorView.frame = CGRectMake(location.x-288 ,location.y-150,
    60,60);
590         _predatorView.alpha = 1;
591         _predatorView.layer.zPosition=99;
592 
593         UIImageView* Imageview = [[UIImageView
    alloc]initWithImage:_pointTrace.trace];
594         Imageview.frame = CGRectMake(location.x-288 ,location.y-150, 60,60);
595         [_backgroundAudio setVolume:0.2];
596         [_pointTrace playSound];
597         [self.drawView addSubview:Imageview];
598 
599         NSMutableArray *toDelete = [[NSMutableArray alloc]init];
600         location = [[touches anyObject] locationInView:imageView1];
601         fingerRect = CGRectMake(location.x-25, location.y-25, 50, 50);
602         for(UIView *point in self.wayPoints){
603             CGRect subviewFrame = point.frame;
604             if(CGRectIntersectsRect(fingerRect, subviewFrame)){
605                 [toDelete addObject:point];
606                 [point removeFromSuperview];
607              }
608         }
609         [self.wayPoints removeObjectsInArray:toDelete];
610 
611         if([self.wayPoints count]==0)
612             [self nextPlanAnimation];
613     }else{
614         [self.wrongPathAudio prepareToPlay];
615         [self.wrongPathAudio play];
616     }
```

Na linha 606 é possível ver que a `presa` é retirada do objeto `drawView`, mas não é removido do `wayPoints` pois esse vetor não pode mudar enquanto estiver dentro da estrutura de *loop*. Foi necessário criar um vetor secundário para armazená-los temporariamente e depois removê-los, o `toDelete` na linha 599.

O método `isWallPixel` trata apenas de verificar se o ponto `x,y` que o `usuário` tocou é um local válido ou não. O Quadro 12 mostra o método em questão.

Quadro 12 – Método `isWallPixel`

```
627 - (BOOL)isWallPixel: (int) x :(int) y {
628     if (x<0||x>imageView1.frame.size.width|| y<0
    ||y>imageView1.frame.size.height) {
629         return false;
630     }
631     const UInt8* data = CFDataGetBytePtr(_pixelData);
632     int pixelInfo = ((imageView1.image.size.width *(y*_scale))+(x*_scale))*4;
633     int alpha = data[pixelInfo + 3];
634     if (alpha==0)
635         return NO;
636     else
637         return YES;
```

Na linha 633 mostra que o *pixel* em questão é referenciado no vetor `data` e armazenado em uma variável local apenas o alpha. O mesmo é testado, caso seja 0 significaria que o ponto está do lado externo da imagem.

O método `touchesEnded` também está presente de forma nativa na classe. Ele reconhece quando `usuário` terminou de tocar na tela. Para esse caso ele foi sobrescrito para fazer a imagem do predador desaparecer, aumentar o volume do áudio de fundo e parar o som do traçado. O Quadro 13 mostra o método em questão.

Quadro 13 – Método `touchesEnded`

```
527 -(void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event{
528     _predatorView.alpha = 0;
530     [_backgroundAudio setVolume:0.7];
531     [_pointTrace stopSound];
532 }
```

O método `nextPlanAnimation` é responsável por executar a animação quando o `usuário` passa por todos os `waypoints`. Ele só é chamado quando o `usuário` termina a `prancha`. O Quadro 14 mostra a sua funcionalidade. Alguns comandos de efeitos foram removidos para melhor entendimento.

Quadro 14 – Método `nextPlanAnimation`

```
702 -(void)nextPlanAnimation{
703     [_backgroundAudio setVolume:0.2];
704     [self.previewView playSoundFromCurrentPlan];
…
714     if([self.previewView isOver]){
715      UILabel* finishLabel = [[UILabel
    alloc]initWithFrame:_backgroundImageView.frame];
716      finishLabel.text = @"Parabéns, plano finalizado!";
717      finishLabel.textAlignment = NSTextAlignmentCenter;
718      finishLabel.font = [UIFont fontWithName:@"times" size:50];
719      [_backgroundAudio setVolume:0.2];
 …
735                 [self.historicView addOnHistoric:_drawView];
736                 self.drawView.transform =
    CGAffineTransformScale(CGAffineTransformIdentity, 1, 1);
737                 self.drawView.frame = imageView1.frame;
738                 [self nextPlan];
739                 [_backgroundAudio setVolume:0.7];
```

A linha 704 mostra que o método executa o áudio da `prancha` atual e depois inicia a animação. Ele faz uma transformação no objeto `drawView` que possui o traçado do `usuário`. Após a animação ser finalizada a transforma novamente para o seu tamanho e posição original. Ao final, chama o método `nextPlan`. A linha 714 exibe a condição para caso não exista mais uma próxima `prancha`. Caso realmente não tenha, uma `UILabel` é exibida na tela com um efeito para mostrar ao `usuário` que o jogo foi acabou.

O método `nextPlan` é responsável por trazer a próxima `prancha` para o centro. Ele é executado diretamente quando o `usuário` toca no botão próximo ou indiretamente quando termina a `prancha` atual. O método pode ser visto no Quadro 15.

Quadro 15 – Método `nextPlan`

```
742 -(void)nextPlan{
743     if(![self.previewView isOver]){
744         [imageView1 setImage:[self.previewView nextPlanOnPreview]];
745 self.pixelData =
    CGDataProviderCopyData(CGImageGetDataProvider(imageView1.image.CGImage));
746         [[_drawView subviews]
    makeObjectsPerformSelector:@selector(removeFromSuperview)];
747         _scale = imageView1.image.size.width/imageView1.frame.size.width;
748         [self makeWayPoints];
749     }
750
751 }
```

Na linha 743 o método verifica se essa `prancha` é a ultima. Se for ele toca o áudio de todas as `pranchas`. Caso contrário a imagem central é trocada para a próxima `prancha`. Todo o traço da antiga `prancha` é removida e as novas presas são posicionadas.

3.3.3 Classe `TGPreviewView`

A classe `TGPreviewView` herda as funções da classe `UIScrollView` e é uma das classes mais importantes. Ela faz o gerenciamento do `plano` do jogo. Apenas essa classe que possui a visão de todas as `pranchas`. Ela tem a sua exibição na parte superior da tela que mostra para o `usuário` as `pranchas` do `plano` e a `prancha` atual que está sendo exibida do centro da tela. O Quadro 16 mostra parte da inicialização da classe.

Quadro 16 – Inicialização da classe `TGPreviewView`

```
23  - (id)initWithPlans:(NSArray*)plans andCurrentPlan: (Plan*) currentPlan
24  {
25      self = [super init];
26      if (self) {
27          self.frame = CGRectMake(170, 45, 800, 80);
28          self.currentPlan = (int)[plans indexOfObject:currentPlan];
29          _symbolPlanController = [[TGSymbolPlanController alloc]init];
30          NSArray* symbolPlansArray;
31
32          for(int i =0; i<[plans count]; i++){
33            Plan* plan = [plans objectAtIndex:i];
34            symbolPlansArray = [_symbolPlanController
    loadSymbolPlansFromPlan:plan];
35            SymbolPlan *symbolPlan = [symbolPlansArray objectAtIndex:0];
36            Symbol *symbolFromPlan = [[[symbolPlan
    symbol]allObjects]objectAtIndex:0];
37            UIImage *image = [UIImage imageWithData:[symbolFromPlan picture]];
38            UIImageView* view = [[UIImageView alloc]initWithImage:image];
39            view.frame =CGRectMake(80*i+10, 0, 80, self.frame.size.height);
40            [self addSubview:view];
41          }
42          self.plans = [[NSMutableArray alloc]initWithArray:plans];
43          self.borderSelected =[[UIImageView alloc]initWithFrame:
    CGRectMake(80*_currentPlan+10, 0, 80, self.frame.size.height)];
44          self.borderSelected.layer.borderColor = [UIColor redColor].CGColor;
45          self.borderSelected.layer.borderWidth = 2;
46          [self addSubview:self.borderSelected];
47
48          _speechSynthesizer = [[AVSpeechSynthesizer alloc]init];
49          _speechVoice = [AVSpeechSynthesisVoice voiceWithLanguage:@"pt-BR"];
50          self.contentSize = CGSizeMake(80*[plans count]+10, 80);
51           [self setScrollEnabled:YES];
```

A partir da linha 32 pode ser visto como é preparada a visão para o `usuário` na parte superior da tela. Cada `símbolo` do `plano` é adicionado em uma `UIImageView` e posicionadas lado a lado. A borda vermelha que indica qual é a `prancha` atual é inserida logo depois e posicionada segundo o valor do `currentPlan`.

O método `refreshPlan` é responsável por retornar a imagem que irá para o centro que será a atual. O Quadro 17 exibe os detalhes do método sendo que os comandos de animações foram removidos para melhor visualização.

Quadro 17 – Método `refreshPlan`

```
71  -(UIImage*)refreshPlan{
72      Plan* plan = [self.plans objectAtIndex:self.currentPlan];
73      NSArray* symbolPlansArray = [_symbolPlanController
    loadSymbolPlansFromPlan:plan];
74      SymbolPlan *symbolPlan = [symbolPlansArray objectAtIndex:0];
75      Symbol *symbolFromPlan = [[[symbolPlan
    symbol]allObjects]objectAtIndex:0];
…
84      return [UIImage imageWithData:[symbolFromPlan picture]];
85  }
```

Tanto para a próxima `prancha` como para a anterior o mesmo método é chamado, pois apenas é atualizado a sua referencia para o vetor de `pranchas`. Como pode ser visto a partir da linha 72 do Quadro 17, o método pega a referência da `prancha` atual, pega o primeiro `símbolo`, que nesse caso será o único e envia a imagem do mesmo.

A classe `TGPreviewView` também é responsável por tocar o áudio de cada `prancha` ao final da mesma e o áudio final do `plano`. No Quadro 18 é possível ver o método `playSoundFromGroupPlan`.

Quadro 18 – Método `playSoundFromGroupPlan`

```
106 -(void)playSoundFromGroupPlan{
107     NSString *word = [[NSString alloc]init];
108     for (int i =0; i< [self.plans count]; i++) {
109         Plan* plan = [self.plans objectAtIndex:i];
110         NSArray* symbolPlansArray = [_symbolPlanController
    loadSymbolPlansFromPlan:plan];
111         SymbolPlan *symbolPlan = [symbolPlansArray objectAtIndex:0];
112         Symbol *symbolFromPlan = [[[symbolPlan
    symbol]allObjects]objectAtIndex:0];
113         word = [word stringByAppendingString:[symbolFromPlan name]];
114     }
115     _speechUtterance = [[AVSpeechUtterance alloc]initWithString:word];
116     [_speechUtterance setVoice:_speechVoice];
117     [_speechUtterance setPitchMultiplier:0.9];
118     [_speechUtterance setRate:AVSpeechUtteranceMinimumSpeechRate];
119     [_speechSynthesizer speakUtterance:_speechUtterance];
```

O método `playSoundFromGroupPlan` que passa por todos os `símbolos` para obter seu nome e concatenar em uma *String* que é reproduzida como áudio final pelo sintetizador de voz. Na linha 19 pode-se ver o comando que faz o sintetizador iniciar a sua fala.

3.3.4 Classe `TGHistoricView`

A classe `TGHistoricView` herda funções da classe `UIScrollView` também. Ela apenas exibe o desenho que o `usuário` fez na tela em tamanho menor e exibindo todas as `pranchas` que foram finalizadas. O método `addOnHistoric` pode ser visto no Quadro 19 que teve os comandos para efetuar os efeitos removidos para melhor visualização.

Quadro 19 – Método `addOnHistoric`

```
29 -(void)addOnHistoric: (UIView*) view{
30     UIView* historicView = [[UIView alloc]initWithFrame:view.frame];
31     [historicView setAlpha:0];
32     for (UIImageView* image in view.subviews ) {
33         UIImageView* imageview = [[UIImageView
    alloc]initWithFrame:image.frame];
34         imageview.image = image.image;
35         [historicView addSubview:imageview];
36     }
37     historicView.transform = CGAffineTransformScale(CGAffineTransformIdentity,
    0.2, 0.2);
38     historicView.frame = CGRectMake(100*self.numberOfItemsInHistoric+10, 0,
    100, self.frame.size.height);
39     [self setContentSize:CGSizeMake(100*self.numberOfItemsInHistoric+10,
    self.frame.size.height)];
40     [self addSubview:historicView];
41     self.numberOfItemsInHistoric++;
```

Esse método é responsável por organizar a nova `UIImageView` recebida. A linha 32 mostra que o método cria uma própria `UIImageView` de mesmo tamanho que a passada por

parâmetro a adiciona todas as `subviews` da original na `UIimageView` local. A partir da linha 37 pode-se ver que essa `view` local é redimensionada e posicionada no devido local.

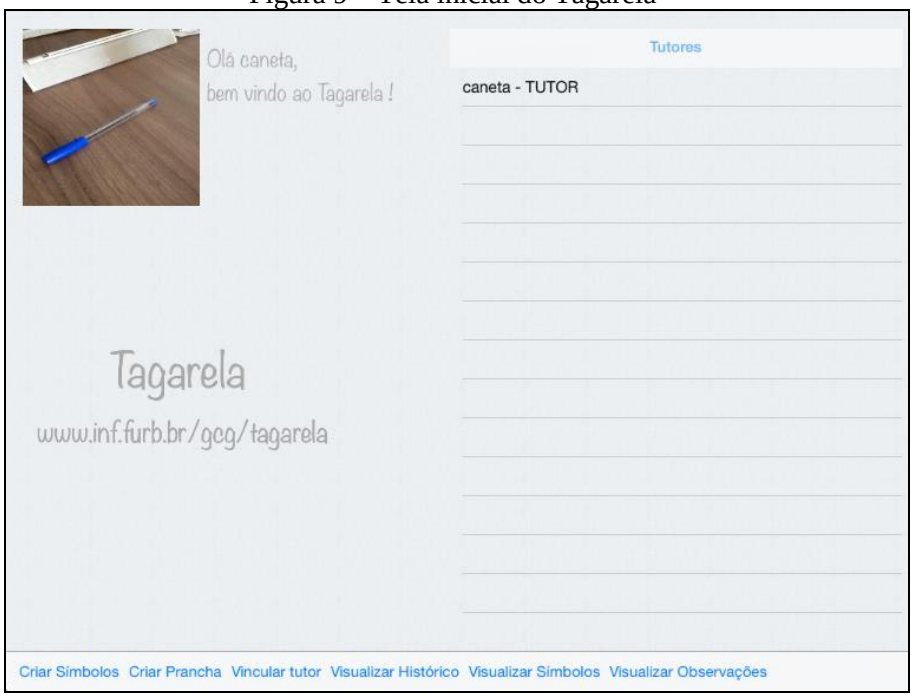
3.3.5 Operacionalidade do jogo

Nessa seção serão demonstradas as principais funcionalidades estendidas do framework Tagarela como a tela inicial, a de criação de `pranchas`, a tela principal e a tela de conclusão do `plano`. Mais detalhes podem ser vistos no trabalho de Fabeni (2012).

3.3.5.1 Tela inicial

Após o `usuário` efetuar o login no aplicativo, a tela inicial do Tagarela é aberta. Ela exibe na lateral direita todos os `planos` disponíveis do `usuário`. Nesta tela o `usuário` poderá criar uma nova `prancha` para adicionar em um `plano` ou jogar um `plano` existente. A Figura 9 exibe a tela inicial do Tagarela.

Figura 9 – Tela inicial do Tagarela



A figura 10 mostra a tela inicial do projeto de Reetz (2013). Como pode ser visto, ele possui um menu para o jogo e o projeto feito para iOS utiliza o menu do próprio Tagarela.

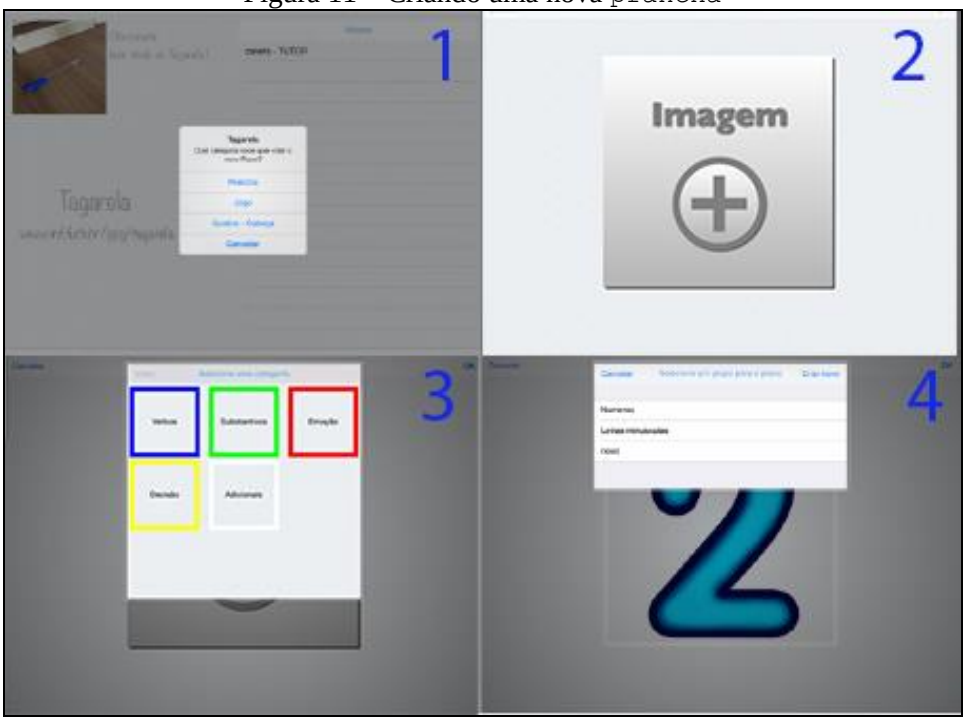
Figura 10 - Tela inicial do jogo para Android

Fonte: Reetz (2013).

3.3.5.2 Criando uma `prancha` para jogo

No momento que o `usuário` toca no botão `criar prancha` uma tela de aviso surge na tela perguntando qual o tipo de `prancha`. Como a Figura 11 (1) mostra, até o momento existem 3 tipos: prancha que é usado na comunicação alternativa, a de jogo que cria pranchas para o desenho em `símbolos` e o quebra-cabeça que ainda não foi feito para a plataforma iOS.

Figura 11 – Criando uma nova `prancha`



Quando o usuário seleciona jogo a tela é recarregada para a criação de uma prancha 1x1 (Figura 11 (2)) onde o usuário poderá selecionar o símbolo que deseja para a prancha. Então ao tocar para inserir o símbolo uma janela com as categorias e seus respectivos símbolos aparecem na tela (Figura 11 (3)). Após o usuário fazer a seleção do símbolo e tocar em ok ele deve selecionar em qual plano ele pretende salvar essa nova prancha. Uma janela com uma lista de todos os planos que são do tipo jogo é exibida (Figura 11 (4)). O usuário pode selecionar uma delas ou criar uma nova. Caso o usuário crie um novo, após dar o nome do novo plano ele é adicionado a lista de planos podendo ser selecionado. Ao final a prancha é adicionada ao plano selecionado. A Figura 11 mostra cronologicamente a criação da nova prancha.

3.3.5.3 Tela principal do jogo

Quando o usuário selecionar um plano que é do tipo jogo a tela para jogar o plano será exibida para o usuário. Como mostra a Figura 12 a tela é dividida em três principais partes, que são: pré-visualização de pranchas (Figura 12 (a)), símbolo principal (Figura 12 (B)) e histórico de pranchas (Figura 12 (C)). A pré-visualização de pranchas localizada na parte superior da tela é a área reservada para o jogo listar todas as pranchas existentes no plano, e também terá função de destacar a prancha atual. O símbolo principal localizado no centro da tela é a área reservada para o Usuário realizar o desenho do símbolo. O histórico de pranchas localizada na parte inferior da tela é a área reservada para o jogo listar as pranchas concluídas pelo Usuário. A parte inferior também possui os botões próximo e anterior que carrega próxima prancha ou a anterior sem precisar desenha a atual. Essa tela também exibe os botões para alterar imagem de fundo, predador, presa e traço. Ao alterar o plano de fundo também se altera o áudio de fundo que é trocado pelo som do símbolo selecionado. Isso também acontece ao trocar o símbolo que representa o traço. Um exemplo da tela principal pode ser visto na Figura 12.

Figura 12 – Tela principal do jogo

O usuário inicia a interação tocando na tela. Para o usuário concluir uma prancha ele precisa passar por todos os pontos destacados por presas no meio do símbolo. Somente dessa maneira a prancha irá para o histórico abaixo do símbolo principal. Não há um ponto de inicio, portanto o usuário pode começar por qualquer ponto. A Figura 13 mostra o símbolo sendo desenhado e o posicionamento do novo símbolo.

Figura 13 – Interação com o símbolo principal

No momento que o usuário concluir o desenho do símbolo o áudio correspondente ao símbolo será tocado e a próxima prancha será carregada. Quando o usuário desenhar

corretamente a última `prancha` o áudio do plano é tocado e o `usuário` recebe uma mensagem de conclusão. A Figura 14 apresenta a mensagem exibida para o `usuário` ao concluir as atividades do plano.

Figura 14 – `Plano` sendo concluído

3.4 RESULTADOS E DISCUSSÕES

Este trabalho apresenta uma protótipo de um jogo 2D de Letras/Números voltado para tecnologia assistiva na plataforma iOS. Nas seções a seguir serão mostrados os resultados de memória e desempenho nas plataformas de testes.

3.4.1 Memória e desempenho

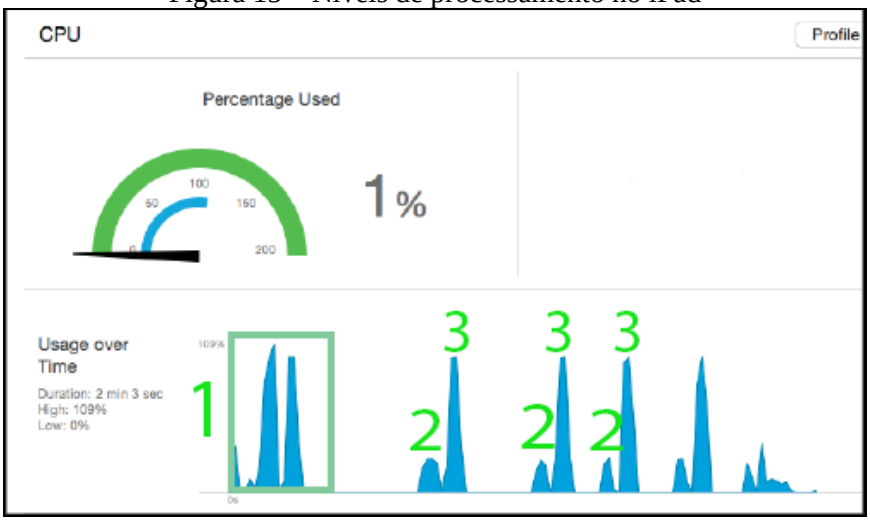
O Xcode exibe alguns gráficos de desempenho da aplicação. Atualmente ele nos permite acompanhar a porcentagem de uso do processador, memória, acesso ao disco e acesso a internet. Para os testes o acompanhamento principal foi do uso do processador e memória. Os testes foram divididos em duas seções: processamento e memória.

Os testes foram feitos em um iPad 2. O iPad 2 tem processador dual-core A5, 512 MB de RAM, com tela de 9,7 polegadas e resolução 768 x 1024 *pixels* e seu sistema operacional é o iOS 7.1.

3.4.1.1 Processamento

No dispositivo físico foram feitos os principais testes para verificar se o jogo poderia consumir muitos recursos. O jogo foi monitorado primeiramente o processamento como pode ser visto na Figura 15 que mostra os níveis alcançados no *tablet*.

Figura 15 – Níveis de processamento no iPad



Após a sincronização inicial do Tagarela `(1)` pode ser visto o processamento durante o desenho `(2)` foi aceitável. Nesse período o jogo atingiu os 25% de processamento máximo, mas no período seguinte `(3)` que acontece a troca de uma nova imagem e ela é percorrida teve períodos de 100%. As transições entre `pranchas` no jogo acabaram ficando lentas, mas nada muito aparente. Lembrando que o processador de um iPad 2 é dual core, fazendo o seu limite de processamento ser 200%. Isso faz com que o aplicativo trave ao carregar, mas mantém funcional o sistema operacional. O Quadro 20 mostra os resultados obtidos juntamente com o tempo que cada função leva para executar.

Quadro 20 – Níveis de processamento durante o jogo

Função	Nível de processamento	Tempo para a finalização (em segundos)
Posicionamento dos waypoints	>101%	2.503
Carregamento inicial	>27%	0.070
Durante o traçado	>26%	0.035

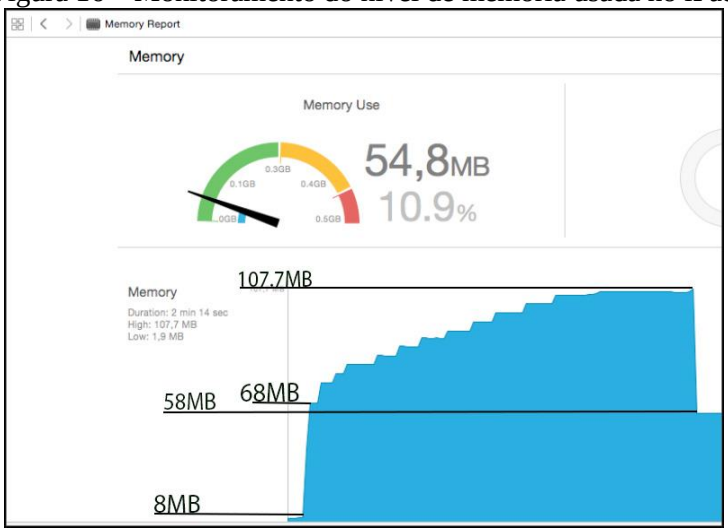
Como pode ser visto o processamento principal é o momento que carrega os pontos de interesse para posicionamento da presa e pode levar mais de dois segundos e meio. E como é para cada prancha isso pode interferir na usabilidade do aplicativo. Entretanto o carregamento inicial consome menos de 27% do processamento de um núcleo do iPad e de forma bem rápida. Sendo quase tão rápido quanto o próprio traçado que usa 0.035 segundos para cada vez que o método é executado.

3.4.1.2 Memória

A memória também foi monitorada da mesma maneira que o processador. Inicialmente foi exibida a alocação de memória do framework para as funções gerais de sincronização e

exibição de telas e depois o jogo foi iniciado. A Figura 16 mostra o nível de memória usado antes, durante e depois do jogo.

Figura 16 – Monitoramento do nível de memória usada no iPad



O framework consome até 25 megabytes de memória no uso das outras funções que não são do jogo. Nesse caso estava com 8MB pois não teve interação antes de entrar no jogo. E ao escolher uma `prancha` do tipo jogo esse valor sobe para aproximadamente 68 megabytes. Isso acontece por causa da `pré-visualização` que precisa carregar todas as `pranchas` do `plano`. Cada pequena elevação que ocorre na memória é uma `prancha` que é concluída. Essa prancha é removida da parte principal e o desenho feito pelo `usuário` é passado para o histórico. Isso faz o jogo alocar aproximadamente 5 megabytes por `prancha` concluída.

É interessante lembrar que a memória vai variar conforme a quantidade de `pranchas` contidas no `plano`. Os `planos` testados continham entre 5 e 10 `pranchas`. Cada `prancha` adicionada ao `plano` somava aproximadamente 4 megabytes ao uso de memória. Lembrando que cada `prancha` ao ser concluída e adicionada ao histórico consome aproximadamente 4 megabytes também. O Quadro 21 exibe os níveis de memória alocada dependendo da quantidade de pranchas.

Quadro 21 – Nível de memória em diferentes quantidades de pranchas

Quantidade de `pranchas`	Quantidade me memória alocada	Memória consumida após o `plano` concluído
1	31MB	32MB
5	44.6MB	61MB
10	68MB	107.7MB
20	108MB	181MB

Em testes com um maior número de `pranchas` a quantidade de memória pode ser significativa. Em `planos` com número de `pranchas` acima de 20 o consumo de memória chegou a 108 megabytes apenas no carregamento inicial. E `planos` com esse tamanho podem existir como, por exemplo, o alfabeto completo. Porém mesmo que esse `plano` seja carregado e totalmente concluído ainda não chegaria ao nível de atenção de um iPad 2 que é de 300 megabytes tornando um `plano` deste porte totalmente possível.

Quando a parte de jogos da aplicação é fechada o nível de memória não retorna ao nível normal, sempre um pouco acima dependendo da quantidade alocada anteriormente. Isso porém ocorre em todo o aplicativo e se continuar a usá-lo por muito tempo o sistema poderá fechá-lo por falta de memória.

3.4.2 Comparativo entre o trabalho desenvolvido e os trabalhos correlatos

Nesta seção é apresentada a comparação entre as principais características deste trabalho com os dos trabalhos correlatos. O Quadro 22 apresenta um comparativo entre este trabalho e os trabalhos correlatos.

Quadro 22 – Comparativo entre os trabalhos

	Jogo 2D desenvolvido	Livox	Desenhe e aprenda a escrever	Reetz 2013
Apelo pedagógico	X	X	X	X
Pranchas e símbolos	X	X	X	X
Planos customizados	X	X	X	X
Gratuito	X			X
Possui planos/cenários nativos			X	X
Permite acentuação das letras	X		X	
Permite a escolha de símbolos para personalização durante o jogo	X		X	

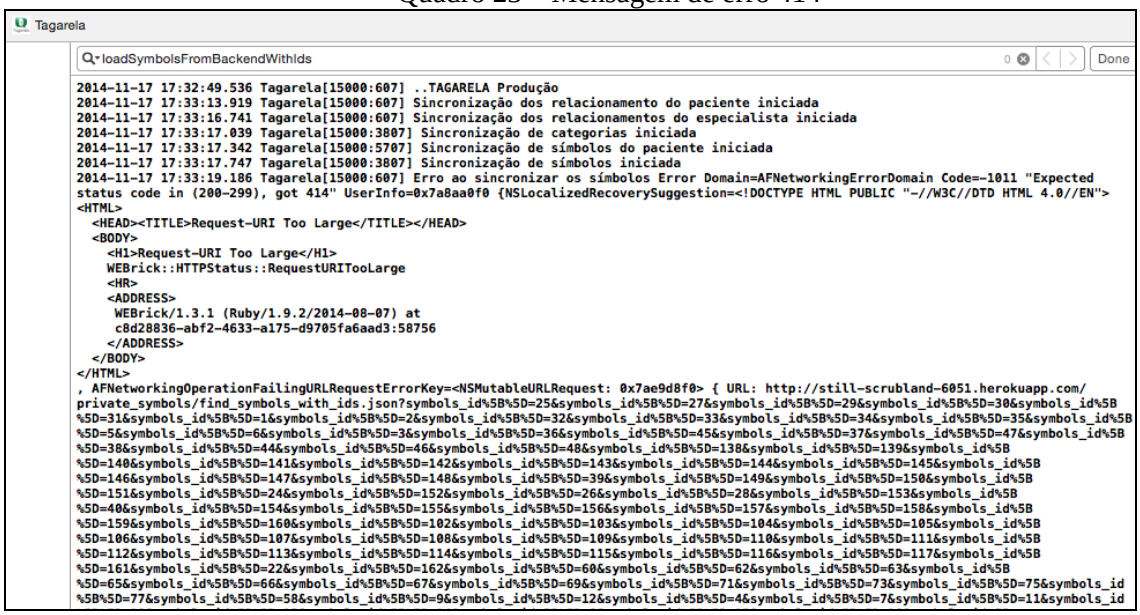
Como pode ser visto no quadro o trabalho pode ser comparado ao trabalho de Reetz (2013) que torna a migração do projeto bem sucedida. O trabalho também pode ser comparado ao Desenhe e Aprenda a Escrever desenvolvido por FizzBrain (2014) pois aceita a

personalização de símbolos no jogo e acentuação das letras. Porém o trabalho desenvolvido permite a sincronização dessas personalizações em outros *tablets* que o torna útil para crianças com necessidades especiais. E como é integrado ao Tagarela (2014) está disponível para os pacientes, tutores e especialistas que usarem o *framework.*

As letras acentuadas foram permitidas pelo próprio aplicativo, ou seja, nenhuma modificação teve de ser feita. O usuário apenas precisa criar um símbolo com a letra acentuada em questão e depois adicionar uma prancha com o símbolo a um plano.

Um problema grave encontrado no Tagarela foi na sincronização do aplicativo. Se o usuário tiver muitos símbolos para sincronizar o aplicativo trava, pois apresenta o erro 414. O Quadro 23 exibe o erro que aparece no console do Xcode após a tentativa de sincronização com o banco de dados.

Quadro 23 – Mensagem de erro 414



O erro 414 é sobre o tamanho da *string* de requisição que está muito grande. O framework tentava sincronizar todos os símbolos de uma vez e isso fazia a requisição HTTP passar do limite permitido. Se o `usuário` já estiver sincronizado em um *tablet* e inserir novos `símbolos` não será problema. O erro acontece quando o `usuário` efetuar o *login* em outro dispositivo. Esse então não conseguirá trazer nenhum `símbolo` da base de dados. O erro foi resolvido fazendo uma requisição para cada símbolo.

4 CONCLUSÕES

Com a reutilização do código de Fabeni (2012) foi possível elaborar o jogo 2D, podendo ser executado em um iPad 2 e sem chegar a metade da capacidade máxima do dispositivo em boa parte da execução do programa.

Um dos objetivos era analisar a possibilidade de adicionar um sintetizador de voz ao final do `plano`. Como a aplicação não demonstrou sinais de lentidão esse objetivo acabou sendo implementado. Porém o sintetizador acaba falando toda a frase do `plano`, mesmo que o `usuário` tenha feito apenas a última `prancha`.

Alguns sinais de lentidão no dispositivo físico foram encontrados no momento de adicionar as presas na imagem. Apesar de ser apenas uma vez por `prancha` isso pode desagradar o `usuário`. A lentidão fica ainda mais evidente quando o `usuário` toca no botão para a próxima `prancha` ou anterior muitas vezes seguidas. Vale ressaltar que o dispositivo em questão foi um iPad 2, considerado ultrapassado pelos padrões de mercado atuais.

Para evitar esse período de lentidão entre pranchas seria interessante pesquisar uma maneira de fazer com que o jogo não use as presas para determinar se a prancha foi concluída ou não. Evitando a necessidade de varrer cada *pixel* da imagem para verificar o alpha para posicionamento das `presas`. A maneira utilizada hoje também faz com que os `símbolos` fiquem incompletos na hora de desenhar. Isso se torna uma maneira fácil de burlar o jogo já que seria necessário apenas capturar todas as `presas` e não fazer o desenho.

4.1 EXTENSÕES

Para trabalhos futuros as seguintes extensões são sugeridas:

a) adicionar ao histórico geral do projeto Tagarela quando o `usuário` concluir uma `prancha` ou um `plano`. Podendo ser visto na tela inicial do aplicativo para melhor acompanhamento;
b) pedir ao `usuário` os pontos de interesse onde ficarão as presas quando o mesmo adicionar uma nova `prancha` ao `plano`;
c) utilizar uma `UICollectionView` ou semelhante na parte de pré-visualização, permitindo ao `usuário` selecionar a `prancha` que ele deseja de maneira intuitiva;
d) pesquisar uma forma de fazer que seja necessário passar por todo o desenho e não apenas pelas `presas` do `símbolo`;
e) adicionar um botão para remover todo o `traçado` feito na `prancha`.

REFERÊNCIAS BIBLIOGRAFICAS

BERSH, Rita C.R.; PELOSI, Miryam B**. Tecnologia assistiva:** recursos de acessibilidade ao computador. Brasília, 2011. Disponível em: <https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnx0ZWNub2xvZ2lhYXNzaXN0aXZhY29tYnJ8Z3g6NTJmNjkzNTIwMTNkYmUwYg> Acesso em: 21 mar. 2014

CORREIA, Patrícia V. A. **Alteração de um paradigma e o futuro da comunicação alternativa: vox4all, um caso de estudo**. Gramado, 2013. Disponível em: <http://www.ufrgs.br/teias/isaac/VCBCAA/pdf/114772_1.pdf>. Acesso em: 13 maio 2014

CORREIA, Vasti G. P. **A comunicação alternativa como via de acesso à inclusão: visibilizando os potenciais comunicativos de uma aluna com paralisia cerebral.** Gramado, 2013. Disponível em: <http://www.ufrgs.br/teias/isaac/VCBCAA/pdf/115897_1.pdf>. Acesso em: 15 maio 2014

CUPERSCHMID, Ana R. M.; HILDEBRAND**,** Hermes R. **Heurísticas de jogabilidade: usabilidade e entretenimento em jogos digitais.** Campinas: Marketing Aumentado**,** 2013.

ESTRELLA, Sérgio et al**. Coleção Nintendo Blast - Ano 1**.Porto Alegre: Gameblast, 2010.

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.

FEIJÓ, Bruno; SILVA, Flávio S. C.; CLUA, Esteban. **Introdução à ciência da computação com jogos.** Rio de Janeiro: Elsevier, 2009.

FIZZBRAIN. **Desenhe e aprenda a escrever.** Los Angeles, 2013. Disponível em: <https://itunes.apple.com/us/app/desenhe-e-aprenda-a-escrever!/id545187337?mt=8&ign-mpt=uo%3D4>. Acesso em: 04 abr. 2014

GEBRAN, Maurício P. **Tecnologias educacionais**. Curitiba: Iesde, 2009.

INSTITUTO DE TECNOLOGIA SOCIAL - ITS BRASIL. **Tecnologia assistiva nas escolas**. Rio de Janeiro, 2008. Disponível em: <https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnx0ZWNub2xvZ2lhYXNzaXN0aXZhY29tYnJ8Z3g6M2NjMmUzN2E0ZjBmODA3Yg>. Acesso em: 21 mar. 2014.

LIVOX. **Livox**. Rio de Janeiro, 2014. Disponível em: <http://www.agoraeuconsigo.org/quem-somos/>. Acesso em: 24 mar. 2014

LORENA, Patrícia Q**. Tecnologia assistiva e comunicação alternativa.** [S.l.], 2010. Disponivel em: <http://www.bengalalegal.com/ca-comunicacao-alternativa>. Acesso em: 17 mar. 2014

REETZ, Wagner F. **Jogo de letras/números voltado para tecnologia assistiva no Android.** 2013. 62 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

TAGARELA. **Tagarela:** plataforma de comunicação alternativa. Blumenau, 2014. Disponível em: <http://gcg.inf.furb.br/tagarela>. Acesso em: 09 abr. 2014.

TAROUCO, Liane at al. **Jogos educacionais.** Ceará, 2004. Disponível em: <http://www.virtual.ufc.br/cursouca/modulo_3/jogos_educacionais.pdf>. Acesso em: 26/05/2014