UNIVERSIDADE FEDERAL DOS VALES JEQUITINHONHA E MUCURI

FACULDADE DE CIÊNCIAS EXATAS
CURSO DE SISTEMAS DE INFORMAÇÃO

MARCUS VINICIUS LOPES ARAUJO

APRENDIZAGEM MÓVEL NO ENSINO SUPERIOR: SUPORTE AO ENSINO DE FUNDAMENTOS DE MATEMÁTICA

Diamantina
2016

MARCUS VINICIUS LOPES ARAUJO

APRENDIZAGEM MÓVEL NO ENSINO SUPERIOR: SUPORTE AO ENSINO DE FUNDAMENTOS DE MATEMÁTICA

Trabalho apresentado ao curso de Sistemas de Informação da Universidade Federal dos Vales Jequitinhonha e Mucuri – UFVJM, como pré-requisito para obtenção do grau de bacharel, sob orientação do Profª. Dra. Luciana Pereira de Assis.

Diamantina
2016

MARCUS VINICIUS LOPES ARAUJO

APRENDIZAGEM MÓVEL NO ENSINO SUPERIOR: SUPORTE AO ENSINO DE FUNDAMENTOS DE MATEMÁTICA

Trabalho apresentado ao curso de Sistemas de Informação da Universidade Federal dos Vales Jequitinhonha e Mucuri – UFVJM, como pré-requisito para obtenção do grau de bacharel, sob orientação do Profª. Dra. Luciana Pereira de Assis.

COMISSÃO EXAMINADORA

_____

Profª. Dra. Luciana Pereira de Assis (Orientadora)
Universidade Federal dos Vales Jequitinhonha e Mucuri

_____

Prof. Dr. Alessandro Vivas Andrade
Universidade Federal dos Vales Jequitinhonha e Mucuri

_____

Prof. Dr. Alex Erickson Ferreira
Universidade Federal dos Vales Jequitinhonha e Mucuri

Diamantina, ____ de_____ de 20_

À todos familiares...

AGRADECIMENTOS

Primeiramente demonstro minha gratidão a Deus e todos os meus amigos espirituais que me deram forças, sabedoria e equilíbrio ao logo de toda minha vida. Agradeço também ao meu pai Neri José, minha mãe Maria Aparecida Nunes, meu irmão Nerimar Dawid e meu irmão Vinícius Moraes, que sempre me apoiaram, demostrando preocupação, carinho e amor a minha pessoa. Agradeço aos demais familiares por todas as boas vibrações que me mandaram me dando equilíbrio nas horas difíceis. Agradeço ao meu amigo de república e da família Rainan Fernandes, por todo apoio e camaradagem e também à minha amiga de república, cunhada e irmã Tatiana Fernandes por toda atenção, carinho e disposição a me ajudar e ouvir nos momentos bons e nos momentos complicados. Agradeço aos meus amigos Carlos Eduardo Guedes, Fernanda Gontijo, Tamara Castro e Thamires Macedo por toda atenção e encorajamento a mim prestados, me auxiliando e aconselhando em momentos turbulentos e por compartilharem as alegrias independente de qualquer desavença. Aos colegas que compartilharam bons momentos. Agradeço ao professor Alex Erickson pelo apoio prestado ao longo da graduação e deste trabalho. Agradeço também, à minha professora e orientadora deste trabalho, Luciana Pereira de Assis, por todo apoio, paciência, ensino prestado e, além de tudo, pela confiança prestada na minha capacidade de aprender ao longo da minha graduação e formação como um futuro profissional da área.

“Onde houver desespero,
que eu leve a esperança
onde houver tristeza,
que eu leve a alegria”

Oração de São Francisco

ARAUJO, Marcus Vinicius Lopes. **Aprendizagem Móvel no Ensino Superior: Suporte ao Ensino de Fundamentos de Matemática.** 2016. 163. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) – Faculdade de Ciências Exatas, Universidade Federal dos Vales Jequitinhonha e Mucuri, Diamantina, 2016.

RESUMO

O presente trabalho propõe o desenvolvimento de um aplicativo para dispositivos móveis com sistema operacional Android. A criação do aplicativo tem como objetivo auxiliar o aprendizado dos alunos na disciplina de Fundamentos de Matemática para os cursos de Sistemas de Informação, Ciências Biológicas e Química da UFVJM com foco no tema Trigonometria. Tal motivação se deu ao fato da dificuldade encontrada por parte dos alunos no aprendizado da Matemática, a ausência de ferramentas auxiliares ao entendimento da matéria, direcionadas para a disciplina de Fundamentos de Matemática da universidade em questão e, como visto em várias trabalhos e estudos voltados a educação, o lúdico é uma ferramenta poderosa para o ensino e um aliado dos professores na sala de aula. Para desenvolver aplicativos móveis da plataforma Android é necessário o conhecimento sobre a linguagem de programação e o ambiente de desenvolvimento que serão utilizados. O aplicativo desenvolvido foi intitulado como *Show da Trigonometria* e permitirá que seus usuários, após um breve cadastro, responda perguntas relacionadas com o tema abordado e consiga ver seu rendimento de acordo com os acertos adquiridos ao responde-las. Esta versão 1.0 do aplicativo *Show da Trigonometria* permite cadastrar um usuário apenas informando o nome o qual o utilizador da aplicação gostaria de ser referenciado pela aplicação. Com o usuário cadastrado é possível responder a uma sequências de perguntas abordando Trigonometria, onde o utilizador da aplicação terá que optar por uma resposta entre quatro alternativas para solucionar o problema proposto. O jogo proposto permite acompanhar o rendimento do jogador adquiridos ao longo de sua interação com o mesmo.

Palavras-chave: Dispositivos Móveis. Fundamentos de Matemática. Jogos Digitais Educacionais

ARAUJO, Marcus Vinicius Lopes. **Aprendizagem Móvel no Ensino Superior: Suporte ao Ensino de Fundamentos de Matemática.** 2016. 163. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) – Faculdade de Ciências Exatas, Universidade Federal dos Vales Jequitinhonha e Mucuri, Diamantina, 2016.

ABSTRACT

The purpose of this study is to develop an application for mobile devices with Android operating system. This application aims to help the student learning at the Foundations of Mathematics discipline for UFVJM courses: Information Systems, Biological Sciences and Chemistry, focusing on Trigonometry theme. Such motivation took the fact of the difficulty faced by the students in the learning of mathematics and the absence of an important tool to understanding the content, targeted for the Foundations of Mathematics discipline of the university in question, as seen in various works and studies aimed education, playful is a powerful tool for teaching and an ally of the teachers in the classroom. To develop mobile applications Android platform is necessary to know about the programming language and development environment to be used. The developed application has been titled as "Show da Trigonometria" and allow your users after a brief registration, answer questions related to the topic and can see its income in accordance with the arrangements acquired to answer them. After registration, the user can answer a question sequences about Trigonometry, where the application user will have to choose an answer from four alternatives proposed to solve the problem. The proposed game allows monitoring the performance of the player acquired during their interaction with it.

.

Key-words: Mobile Devices. Foundations of Mathematics. Educational Digital Games

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Android Development Tools
AVD	Application Programming Interface
FUVEST	Faculdade Universitária para o Vestibular
IDC	International Data Corporation
IDE	Integrated Development Environment
INPC	Índice Nacional de Preço ao Consumidor
JDK	Java Development Kit
JVM	Java Virtual Machine
MIT	Massachusette Institute of Technology
OHA	Open Handset Alliance
SDK	Software Development Kit
UFF	Universidade Federal Fluminense
UFMS	Universidade Federal do Mato Grosso do Sul
UNIPAMPA	Universidade Federal do PAMPA
UFVJM	Universidade Federal dos Vales Jequitinhonha e Mucuri
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

A presente monografia aborda o desenvolvimento de um aplicativo móvel capaz de auxiliar o aprendizado da Matemática. Inicialmente, na Seção 1.1, será apresentado a motivação para o desenvolvimento de um aplicativo para a disciplina de Fundamentos de Matemática presente na grade curricular dos cursos de Ciências Biológicas, Química e Sistemas de Informação da Universidade Federal dos Vales do Jequitinhonha e Mucuri. Na Seção 1.2 são apresentados os objetivos a serem alcançados com o trabalho e a Seção 1.3 apresentará a estrutura do trabalho.

1.1 Apresentação

“Etimologicamente a palavra JOGO vem do latim LOCUS, que significa gracejo, zombaria e que foi empregada no lugar de ludus: brinquedo, jogo, divertimento, passatempo” (RANGEL, 2009, p. 27).

Os jogos vem de muitos e muitos anos como uma forma de interagir e divertir as pessoas, inicialmente eram criados meramente para agradar as pessoas que assistiam as participações de outras, mas com o tempo as atividades lúdicas se tornou uma forma de se desafiar como um teste das próprias habilidades. A Figura 1.1 mostra um jogo de tabuleiro feito de pedra usado na Roma Antiga e a Figura 2.1, mostra a evolução da forma de jogar, um vídeo game portátil.

Figura 1.1 – Jogo de Tabuleiro

Fonte: Site O Império Romano[1].

[1] Disponível em: <http://imperioromano-marius70.blogspot.com.br/2007/06/o-tabuleiro.html>. Acesso em Janeiro de 2016

Figura 1.2 – PlayStation Portátil

Fonte: Site GameSpot[2].

Conforme Piaget citado por (Wadsworth, 1984, p. 44):

> O jogo lúdico é formado por um conjunto linguístico que funciona dentro de um contexto social; possui um sistema de regras e se constitui de um objeto simbólico que designa também um fenômeno. Portanto, permite ao educando a identificação de um sistema de regras que permite uma estrutura sequencial que especifica a sua moralidade.

Os jogos muitas das vezes são gerados para despertar várias emoções a quem interage com ele, trazendo prazer, alegria, adrenalina e até mesmo a frustação de quem é altamente competitivo ao enfrentar um revés.

Desde criança as pessoas adquirem habilidades e conhecimento praticando jogos, levando esse aprendizado para circunstancias do dia-a-dia. Segundo Orso (1999, p. 7) “a acriança precisa ser alguém que joga para que, mais tarde, saiba ser alguém que age, convivendo sadiamente com as regras do jogo da vida. Saber ganhar e perder deveria acompanhar a todos sempre”.

Os métodos convencionais de ensino como, usar a lousa para passar o plano de oferta das disciplinas ou praticar exercícios no caderno sem auxílio ou até mesmo motivação, muitas das vezes, não é o suficiente para a formação do aprendizado em muitos dos alunos. Com isso jogos são desenvolvidos como uma forma de auxílio a absorção e formação de conhecimento.

Com intuito de absorver aprendizado, os jogos, hoje em dia, são muito explorados para auxiliar o ensino em várias disciplinas, testando a compreensão do jogador em determinada área de conhecimento e dando assistência no estudo e na

[2] Disponível em: <http://www.gamespot.com/articles/metal-gear-solid-peace-walker-delayed-to-june-8-psp-pack-unveiled/1100-6253663/>. Acesso em Janeiro de 2016

formação do saber. Isso faz com que os jogos, junto à tecnologia, sejam explorados cada vez mais nas salas de aula, por prender mais a atenção dos alunos e por ser uma didática mais atrativa. Na Figura 1.3 pode-se notar o uso da lousa como metodologia de ensino e na Figura 1.4 uma sala de aula onde se usa computadores para o aprendizado.

Figura 1.3 – Professor usando a lousa para o ensino

Fonte: Site Educar Para Crescer[3].

Figura 1.4 – Sala de aula com o uso de computadores

Fonte: Blog Manuel Dutra[4].

Segundo Fantacholi (2009):

> Na educação de modo geral, e principalmente na Educação Infantil o brincar é um potente veículo de aprendizagem experiencial, visto que permite, através do lúdico, vivenciar a aprendizagem como processo social. A

[3] Disponível em: < http://educarparacrescer.abril.com.br/blog/isto-da-certo/categoria/professor/ >. Acesso em Janeiro de 2016
[4] Disponível em: <http://blogmanueldutra.blogspot.com.br/2014_02_01_archive.html>. Acesso em Janeiro de 2016

> proposta do lúdico é promover uma alfabetização significativa na prática educacional, é incorporar o conhecimento através das características do conhecimento do mundo. O lúdico promove o rendimento escolar além do conhecimento, oralidade, pensamento e o sentido.

Nota-se, a cada dia mais, que as pessoas estão envolvidas com as tecnologias, em qualquer ambiente que elas estejam inseridas, como, o entretenimento, realização de trabalhos, pesquisas educacionais e interação com outras pessoas, sendo feito essas práticas através de computadores, vídeo games, smartphones entre outros dispositivos. Com isso jogos digitais são explorados na área da educação por ter se tornado uma forma mais acessível por muitos indivíduos inseridos na sociedade, fazendo com que as pessoas consigam aprender de uma forma mais intuitiva e prazerosa por utilizar algo tão comum no seu dia-a-dia e considerado por muitos, objetos para obter conhecimento e ter práticas de lazer.

Para ressaltar as dificuldades da aprendizagem da matemática, Santos, França e Santos (2007) realizaram um estudo e levantaram dados sobre o aproveitamento das pessoas ao realizarem exames envolvendo tal disciplina. Um dos levantamentos foi sobre a prova dos vestibulares da FUVEST[5], onde monstra que, baseada em cima dos pontos das questões de Matemática, a média do vestibular aplicado em 2006 foi de 3.7 em 12 pontos disponíveis, sendo o total de 170.474 candidatos, dos quais 12.016 estiveram ausentes. Já em 2005, a média foi de 4.1 em 11 pontos disponíveis, sendo o total de 154.514 candidatos, dos quais 6.547 estiveram ausentes.

Rodrigues, Vaz e Oliveira (2014) também fizeram um estudo do desempenho dos alunos envolvendo a área de ciências exatas, com intuito de verificar a eficiência de se aplicar o nivelamento em matemática aos alunos da Universidade Federal do Pampa (UNIPAMPA). Foram aplicados aos alunos uma avaliação diagnóstica com perguntas qualitativas e quantitativas a fim de conhecer as principais dificuldades dos alunos da universidade na área e para identificar o desenvolvimentos do mesmo nas habilidades matemáticas. Com o resultado das avaliações em mãos pode-se direcionar melhor o Curso de Nivelamento.

Neste estudo, foram aplicadas avaliações e obteve-se a porcentagem de acerto antes da realização do Curso de Nivelamento. Para o tema Figuras Geométricas e

[5] Fundação Universitária para o Vestibular. Site disponível em: < http://www.fuvest.br/b/chamada.php?anofuv=2016&chamada=2/>. Acesso em Novembro de 2015.

Trigonométricas obteve-se uma porcentagem de apenas 5,26% de acerto, já para a área de Divisão e Fração obteve-se o desempenho de 40,79% de acertos, sendo o tema com maior porcentagem. Os autores ressaltam a grande reprovação em disciplina da área de ciências exatas e que, mesmo o Curso de Nivelamento mostrando melhoria no desempenho dos alunos, a evasão dos discentes deste curso e os desempenhos adquiridos por eles inicialmente ao ingressar na universidade, são preocupantes.

Para saber o rendimento dos alunos da disciplina Fundamentos de Matemática da UFVJM (Universidade Federal dos Vales Jequitinhonha e Mucuri), foi realizado uma entrevista com o professor atual da disciplina. Como citado pelo professor, a disciplina é oferecida a três cursos da universidade, Sistemas de Informação (Bacharelado), Quimica (Licenciatura) e Ciências Biológicas (Licenciatura). O professor ainda informou que leciona a disciplina já há três semestres consecutivos, sendo o último ainda em andamento. O mesmo fez questão de solicitar ao Departamento de Matemática e Estatística da universidade para que lecionasse a disciplina com o intuito de verificar o nível de conhecimento de matemática dos estudantes ao ingressar na mesma, sendo que a disciplina é ofertada no primeiro período dos três cursos e há um grande índice de reprovação e desespero (conforme palavras do professor) dos alunos nesta disciplina.

Através do levantamento feito pelo professor nos dois semestres já concluídos, 2014/2 e 2015/1, que o mesmo lecionou a disciplina, foi notado que no semestre de 2014/2 houve um índice de reprovação de 46,6% dos alunos na turma de Ciências Biológicas, 60,1% de reprovação na turma de Quimica e 61,4% na turma de Sistemas de Informação. Já no semestre de 2015/1 houve um índice de reprovação de 49,6% dos alunos na turma de Ciências Biológicas, 81,4% de reprovação na turma de Quimica e 57,4% na turma de Sistemas de Informação. O professor ressalva que para tal levantamento não foram levados em consideração o número de desistentes ou os que obtiveram aprovação por meio do *Exame Especial*.

O professor ainda informou que os alunos de Ciências Biológicas geralmente desistem da disciplina Fundamentos de Matemática ainda com o semestre em andamento, por motivo da atenção maior prestada por eles pelas disciplinas específicas ao curso e, como a disciplina de Fundamentos de Matemática não é pré-requisito de nenhuma outra matéria em tal curso, os alunos deixam para cursa-la quando estão prestes a formar. Já os alunos de Química geralmente não só desistem

da disciplina como também do curso pelo alto nível de dificuldade encontrado, também, em Fundamentos de Matemática.

Visando a dificuldade e resistência de muitos alunos em aprender e se dedicar à disciplinas com base na Matemática e, como ressaltado anteriormente, o mau rendimento dos discentes da disciplina Fundamentos de Matemática da UFVJM, tendo conhecimento que o jogo é uma forma muito didática de fornecer aprendizado e que o meio digital está cada vez mais presente do cotidiano de vários alunos, este trabalho terá como função primordial desenvolver um aplicativo para dispositivos móveis, chamado de *Show da Trigonometria*. Tal aplicativo terá como foco o auxílio às pessoas na absorção do conhecimento envolvendo a Matemática e o mesmo poderá ser utilizado inicialmente pela disciplina de Fundamentos de Matemática da UFVJM, para que os alunos envolvidos consigam ter maior rendimento ao realizar as atividades avaliativas e obtenham melhores resultados ao vivenciarem o conteúdo proposto pelo docente da disciplina em questão.

A primeira versão do aplicativo proposta neste trabalho irá focar em um tema lecionado aos alunos da disciplina Fundamentos de Matemática da UFVJM. Para determina qual tema seria abordado pela aplicação foi perguntado ao professor atual desta disciplina, ainda na entrevista feita a ele, qual o aproveitamento dos alunos em cada um dos temas/provas aplicado na disciplina. Os resultados podem ser visto pela Tabela 1.1, onde tal tabela mostra as médias de acertos obtidas em avaliações de 20 pontos. O professor deixa claro que os temas fazem parte de uma divisão natural que é estabelecido no plano de curso e informa que “está em estudo uma nova metodologia de cálculo para este índice, portanto não se deve atribuir a estes atuais outra interpretação senão um “mero reflexo” do tema em que os alunos tem melhor (ou pior) afinidade”. A entrevista completa com o professor está anexo, podendo ser encontrada na seção ANEXO A.

A pior média de acertos obtidos, como demonstra a Tabela 1.1, foi no primeiro semestre de 2015, onde os alunos do curso de Química obtiveram uma média de 2,05 acertos em uma prova de 20 questões relacionadas ao tema Trigonometria. Nota-se, também analisando a tabela, que as médias de acertos em tal tema em quase todos os casos são as menores, determinada pelo próprio professor da disciplina como médias muitos ruins.

Tabela 1.1 – Médias de acertos obtidos nas provas da disciplina de Fundamentos de Matemática lecionada na UFVJM

Curso	*Período*	*Funções polinomiais*	*Exponencial e Logaritmo*	*Trigonometria*
Ciências Biológicas	Sem. 2014/2	6,85	5,33	3,68
	Sem. 2015/1	5	7,1	6,3
Química	Sem. 2014/2	6	4	2,4
	Sem. 2015/1	3,3	3,2	2,05
Sist. Informação	Sem. 2014/2	5,42	6	4,39
	Sem. 2015/1	5,7	6,7	7,3

Com essas informações o trabalho proposto aborda o tema Trigonometria, sendo ressaltado pelo professor atual da disciplina que a aplicação a ser desenvolvida é uma ótima ideia, sendo vista por ele com simpatia e, também, como um aliado para melhorar o aprendizado. O mesmo deixa claro que o aplicativo ainda assim não seria a solução para uma questão que tem uma dimensão gigantesca, mas que, toda tentativa que visa atacar o problema de forma construtiva é válida. O professor ainda diz que, o estudo de tal tema a ser abordado pela aplicação também serve para resolver problemas do dia a dia e pode ser utilizado numa abordagem de função para modelar fenômenos que possuem comportamentos periódico/sistemático, como as ondas sonoras ou estudos da ótica clássica.

1.2 Objetivos

O objetivo principal deste trabalho é o desenvolvimento de um aplicativo para dispositivos móveis para a plataforma Android, voltado para a área educacional, que permita oferecer aos estudantes da UFVJM que cursam a disciplina de Fundamentos de Matemática e demais interessados no tema, uma ferramenta que facilite o estudo voltado para a área da Trigonometria. O jogo é baseado em exercícios nos quais possibilitará ao aluno aprender os princípios básicos do tema envolvido para auxiliar na resolução da atividades propostas. Os grupos de exercícios avançam com diferentes conteúdos da Trigonometria.

Com o intuito de alcançar esse objetivo serão necessários os seguintes objetivos:

- Pesquisar e aprender a linguagem de programação Java, voltada para

dispositivos móveis, em especifico para o sistema operacional Android, para que permita que o aplicativo funcione corretamente nos mais variados dispositivos.

- Utilizar e estudar metodologias e jogos mais didáticos e intuitivos para aprendizado.
- Propor e implementar um aplicativo móvel para auxiliar o aprendizado em Matemática.

1.3 Organização do trabalho

O Capítulo 2 apresenta a revisão bibliográfica conduzida no presente trabalho, apresentando jogos relacionados ao auxílio na educação e a influência dos jogos digitais nesse âmbito, tendo como o foco maior a área da Matemática. Também é abordado sobre a evolução dos dispositivos móveis e sua importância. Por fim, são dados exemplos de ferramentas digitais que envolve o ensino da Matemática de uma forma lúdica.

No Capítulo 3 é abordado sobra o metodologia ágil de desenvolvimento Scrum, utilizada para desenvolver a aplicação, também é explanados sobre as linguagens e as ferramentas utilizadas para desenvolvimento do aplicativo *Show da Trigonometria*.

O Capítulo 4 apresenta o aplicativo desenvolvido, mostrando detalhadamente os procedimentos para a implementação da aplicação proposta por esse trabalho.

Por fim, o Capítulo 5 apresenta a conclusão do trabalho proposto e a descrições dos trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta a revisão bibliográfica conduzida no presente trabalho. Inicialmente, na Seção 2.1, é abordado sobre jogos educacionais. A Seção 2.2 apresenta a definição e as influências dos jogos digitais na educação. Na Seção 2.3 é explanado sobre a importância e evolução dos dispositivos móveis. Já na Seção 2.4 é explanado sobre o ensino da matemática de uma forma lúdica e usando ferramentas digitais. Por fim, na Seção 2.5 são apresentados exemplos de jogos e atividades digitais que abordam a área da Matemática.

2.1 Jogos Educacionais

Segundo Okamoto (2011) o jogo traz várias sensações diferentes a uma diversidade de pessoas independentemente de sua localização e cultura, sendo ele uma atividade de difícil explicação e definição. Antigamente, com muito mais facilidade do que nos dias de hoje, as crianças e adolescentes tinham mais liberdade, em relação a segurança pessoal, de poder praticar atividades lúdicas fora de casa, fazendo das brincadeiras e jogos um momento importante de aprendizado para suas vidas.

Okamoto também afirma que os jogos, atualmente, podem ser realizados em diversos ambientes de convivência, sendo praticado com diferentes objetivos, sendo um dos mais importantes, a utilização como instrumento de ensino nas escolas. A autora atesta que os jogos populares, definido por ela como “patrimônio da cultura popular, pois, são do conhecimento e prática de diferentes povos, seus criadores são desconhecidos, e sua transmissão pode acontecer de forma voluntaria” (OKAMOTO, 2011, p. 5), podem ser encontrados na escola tanto como conteúdo a ser ensinado ou até mesmo como uma estratégia de ensino.

De acordo com o trabalho feito por Aguiar (2006), o jogo não se trata apenas de uma prática de lazer mas também como uma ferramenta de aprendizado, se tornando um desafio válido por imergir em seu universo lúdico pessoas com culturas diferentes. Os estudantes carregam consigo, para as instituições de ensino, suas ideias, seus conhecimentos e intuições adquiridas em seu ambiente sociocultural.

Aguiar ainda cita que os jogos não só trazem uma vivencia de uma determinada situação, mas como também, ensinam a trabalhar com símbolos e compreender

analogias. A autora alega que “o jogo passa a ter capacidade de desenvolver potencialidades, habilidades, estímulo de raciocínio e reflexão nos educandos, sendo de fundamental importância para o desenvolvimento integral dos mesmos” (AGUIAR, 2006), fazendo dessa atividade lúdica uma forma de aprendizagem que torna a aula menos cansativa e monótona, evitando a insatisfação de educandos e educadores.

Segundo Ribeiro, Ribeiro e Junior (2007), os professores devem buscar novas alternativas e metodologias de ensino para aliviar a tensão de aprender o conteúdo. Com isso o jogo pode ser uma poderosa ferramenta de educação. Com ele o aluno aprende inconscientemente, pois o ato de se divertir alivia a pressão que os alunos sentem ao se obrigarem a aprender o que lhe exigem e, interagindo e descontraindo ao aprender, os resultados são mais expressivos.

Segundo Dallabona e Mendes (2004, p. 110):

> Educar é um ato consciente e planejado, é tornar o indivíduo consciente, engajado e feliz no mundo. É seduzir os seres humanos para o prazer de conhecer. É resgatar o verdadeiro sentido da palavra “escola”, local de alegria, prazer intelectual, satisfação e desenvolvimento.

Professores usam jogos em computadores e jogos em estilo tabuleiro para passar conhecimento e conseguir ganhar de uma forma mais prazerosa a atenção do aluno, como mostra a Figura 2.1 e a Figura 2.2, respectivamente.

Figura 2.1 – Jogos em computadores usados em sala de aula

Fonte: Site da Escola Curumim[6].

Figura 2.2 – Jogos usados em sala de aula como método de ensino

[6] Disponível em: <http://escolacurumim.webnode.com.br>. Acesso em Janeiro de 2016

Fonte: Site Tiago Aquino[7].

2.2 Jogos Digitais e Jogos Digitais Educacionais

Silva et al. (2009, p.2) afirmam que jogos digitais “são programas executados em plataformas microprocessadas que possuem como primeiro objetivo o entretenimento de seus usuários”. Os autores trazem que a interação entre o indivíduo e o sistema é dada seguindo regras e limitações, fazendo com que o modo de como o jogador participa do jogo depende da plataforma utilizada para a execução do programa.

Os autores também alegam que:

> Durante o jogo, a atividade de jogar pode ser resumida como um processo contínuo de tomada de decisões a partir da avaliação, pelo jogador, de um determinado estado informado pelo programa, até que a atividade seja finalizada por iniciativa do jogador ou pelo programa do jogo. (SILVA et al., 2009, p.3)

De acordo com Machado (2006), os jogos digitais fez com que a indústria de entretenimento crescesse entre as demais e tivesse uma grande evolução financeira. Que o consumo de jogos eletrônicos está se tornado cada vez mais crescente ao passar dos anos, por haver uma grande avanço da tecnologia de informação e do comércio do lazer digital e explicado tanto pela expansão do consumo e pelo fato de vários jovens manterem o hábito de jogar de vários jovens manterem até a idade

[7] Disponível em: < http://www.tiagoaquinopacoca.com.br/palestras/jogos-criativos-para-sala-de-aula/>. Acesso em Janeiro de 2016

adulta.

Segundo Savi e Ulbricht (2008, p.1):

> As instituições de ensino estão ampliando o uso das tecnologias de informação e comunicação para oferecer aos alunos mídias interativas que possam enriquecer as aulas. Os jogos digitais aparecem nesse contexto como um recurso didático que contém características que podem trazer uma série de benefícios para as práticas de ensino e aprendizagem.

Para Monteiro (2007), com o crescimento das tecnologias de informação e comunicação, as instituições de ensino devem se aliar a essa modernização. Apesar da forma de se jogar tem se tornado a cada dia mais virtual ela não perde a sua eficiência de educar. A autora alega que aplicação do meio virtual para educação faz com que o indivíduo não só receba o aprendizado mas também interaja com a forma de aprender, afirmando que “o velho receptor deixa de ser aquele que deve apenas aceitar ou não a mensagem proposta pelo professor para tornar-se sujeito da própria educação numa comunidade educacional interativa” (MONTEIRO, 2007, p.10).

Os jogos educativos computacionais se tornou atualmente uma forma de se obter conhecimento deixando de ser apenas para o lazer sem acréscimos intelectuais, são ferramentas que facilitam o processo de ensino e aprendizagem por serem prazerosos e considerados um meio de diversão, fazendo com que várias instituições repensem a forma que lecionam as mais diversas disciplinas. Segundo Grubel e Bez (2006, p.1) “os jogos educativos tanto computacionais como outros são, com certeza, recursos riquíssimos para desenvolver o conhecimento e habilidades se bem elaborados e explorados”.

2.3 Dispositivos Móveis

Alcantara e Vieira (2010, p.2) definem Tecnologia Móvel “como a forma de acessar a internet e outros recursos computacionais por meio de dispositivos móveis, tais como, celulares, iPhone, iPod, iPad, notebooks, smartpads, dentre outros”. As pessoas estão a cada dia mais interessadas com a gama de facilidade que os dispositivos móveis oferecem, como a própria mobilidade, a facilidade de obter informações independente do lugar, com amplo alcance a qualquer hora, a interação

com outras pessoas e ter acessos a produtos e serviços personalizados.

Os autores ainda mostram que, com a evolução dos dispositivos móveis, há diversas usabilidades que facilitam à vida de seus usuários. As pessoas que viajam podem ter a ajuda dos serviços de localização em sistema GPS ou Global Positioning System (Sistema de Posicionamento Global), mostrando ao usuário sua localização. Outra utilidade é poder fazer compras sem ter de sair do local onde está, em lojas virtuais.

Para ilustrar a evolução dos dispositivos móveis, a Figura 2.3 traz um dos primeiros aparelhos telefônicos, criado no ano de 1943, usado somente para ligações telefônicas, e um Iphone 6, lançado em 2014, que, além de também ser um aparelho celular, traz consigo várias outras funcionalidades, como navegação na Internet, GPS, possibilidades de interação com jogos através de movimentos e reprodutores de mídias.

Figura 2.3 – Um dos primeiros aparelhos telefônicos e o Iphone 6

Os dispositivos móveis, atualmente, são produzidos em larga escala, fazendo assim, que seu preço seja acessível e que seja um objeto da maioria das pessoas, se tornando uma poderosa ferramenta de comunicação e obtenção de informações em tempo real e diferentes lugares.

Segundo Saboia, Vargas e Viva (2013, p.8):

> As tecnologias móveis têm possibilitado que o processo de comunicação e a difusão da informação ocorram em diferentes espaços e tempos, sendo duas de suas características a portabilidade e a instantaneidade. Características que permitem a uma grande parcela da população o acesso a informação em qualquer lugar e a qualquer tempo, seja em tempo real ou não.

Atualmente, crianças e adolescentes passam muito tempo manuseando dispositivos móveis, prática que lhe permite ter acesso a um universo imenso de aprendizado e informações. Segundo Mousquer e Rolim [200-] essa ferramenta computacional permite que o usuário conheça diversificados espaços virtuais e ambientes de conhecimento, sendo bem conduzida traz grandes ganhos ao educando.

Considerando que a tecnologia computacional nas instituições de ensino estimula o desenvolvimento da criatividade, autonomia e socialização, Mousquer e Rolim (200-, p.2) afirmam que:

> [...] o uso de dispositivos móveis como Smartphones, PDAs e Tablets pode abrir muitas oportunidades do aluno trabalhar a sua criatividade, ao mesmo tempo em que se torna um elemento de motivação e colaboração, uma vez que o processo de aprendizagem da criança se torna, atraente, divertido, significativo e auxilia na resolução de problemas que podem ser resolvidos conjuntamente com outras crianças.

A Figura 2.4, mostra que o Ipad, dispositivo móvel da empresa Apple, pode ser uma ferramenta educacional, dando o acesso a diversos livros digitais.

Figura 2.4 – Ipad sendo usado como uma ferramenta educacional

Fonte: Site da Apple[8].

[8] Disponível em: <https://www.apple.com/br/education/ipad/it/mdm/>. Acesso em Janeiro de 2016

2.4 O ensino da Matemática através de jogos e ferramentas digitais

Um dos motivos para se aplicar jogos nas aulas da disciplina da matemática é quebrar o bloqueio e a tensão encontrados por muitos alunos ao se deparar com essa área de estudo, indo ao encontro dela já desmotivados se sentindo incapacitados de aprendê-la.

> Dentro da situação do jogo, onde é impossível uma atitude passiva e a motivação é grande, notamos que ao mesmo tempo em que estes alunos falam da Matemática, apresentam também um melhor desempenho e atitudes mais positivas frente a seus processos de aprendizagem. (BORIN, 1996. p. 09)

Para levar os conceitos de Trigonometria aos alunos de ensino médio, Freire (2000) apresenta e ensina a manusear, alternativas de uso digital para o ensino da Trigonometria. Uma dessas alternativas é o uso de um programa para construir gráficos, conhecido como Funtrig. A autora atesta que o intuito é “preparar o caminho para uma discussão mais rigorosa do relacionamento entre os conceitos de frequência, período, amplitude e deslocamento” (FREIRE, 2000, p.11), trabalhados com Funções Trigonométricas. O uso dessa ferramenta auxilia os alunos a se interessar mais pela área, pois, desenhar e trabalhar com gráficos trigonométricos de forma manual em papel e outros objetos para desenho a mão desestimula e se torna cansativo para muitos aprendizes.

Campos e Novais (2010), em um estudo planejado aborda jogos que fazem vivenciar a probabilidade, onde essas atividades lúdicas levantam questões que estimulam a reflexão e o aprendizado dessa disciplina. Um desses jogos citados pelos autores, chamado de “Jogo dos palitos”, trata-se de dois jogadores, cada um com três palitos, apostam qual será a soma de palitos que serão mostrados por eles, de forma que cada participante escolherá ocultamente quantos palitos dos três irá mostrar. Depois da aposta feita, abre-se a mão e contabiliza a soma da quantidade de palitos escolhidos pelos dois jogadores. Aquele que acertou a soma deve jogar a próxima rodada com um palito a menos. A cada rodada é feita as devidas anotações dos valores das apostas e quem acertou. O vencedor será aquele que alcançar a meta de não ter mais nenhuma palito na mão.

Esse jogo, segundo este autores, “tem como objetivo possibilitar

conhecimentos das chances de ganhos em jogos, introduzir noções probabilísticas, construir tabela de dupla entrada e representação gráfica.” (CAMPOS E NOVAIS, 2010, p. 5). Essa atividade lúdica, levanta questões como “Qual a probabilidade de acontecer a maior soma?”, “Qual a chance de acontecer a menor soma?” e “Qual soma tem maior chance de acontecer?”, estimulando com isso, o raciocínio da disciplina dada como objetivo.

Outros autores também trazem os efeitos dos jogos envolvendo probabilidade nas salas de aula, como o autor Lopes (2008) em “Uma Proposta para o Estudo de Conceitos Básicos de Probabilidade” mostra um estudo feito em uma escola pública do interior de São Paulo, sobre um jogo probabilístico que consiste em os alunos fazerem lançamentos de dois dados para obter certa pontuação que as regras do jogos estipula, ondem devem escolher fazerem novos lançamentos ou não de acordo com a probabilidade de conseguirem mais ou menos pontos. O autor traz a brincadeira no ambiente educacional com o objetivo de ensinar probabilidade através da resolução de problemas.

Este autor também afirma que “o jogo deve ser olhado como um elemento que pode disparar o processo de construção do conhecimento e deve expressar aspectos-chave do tópico matemático que se deseja estudar. Assim, o jogo é utilizado como um ponto de partida e um meio para se ensinar matemática” (LOPES, 2008, p. 1061).

O site PORVIR em uma das suas matérias postadas, fala sobre um jogo digital chamado The Radix Endeavor desenvolvido por pesquisadores MIT (Massachusettes Institute of Technology). Esse jogo usa a lógica dos populares games de estratégia. O site cita que o jogo “propõe estimular práticas de pensamento crítico, criação, colaboração e resolução de problemas.” O jogo envolve os assuntos de álgebra, geometria, probabilidade, estatística e biologia. O jogo foi criado no intuito de deixar o aprendizado das disciplinas tratados por ele mais atraente e estimulante, fazendo com que os alunos não considere o comum e não tão envolvente, sendo que esse permite o modo *online*, tendo assim a socialização com outros jogadores (PORVIR, 2013).

As disciplinas de Exatas, como Fundamentos de Matemática, são vistas pelos alunos como dificuldade e empecilho para alcançar seus objetivos educacionais, como concluir o curso. Com isso há estudos e pesquisas para criação de jogos para fazer com que o processo de aprendizado se torne prazeroso, tendo como um ponto importante a análise feita pelo educador dos benefícios e da eficácia do jogo, sendo ele útil ou não para a aplicação. Logo, este trabalho também apresenta uma forma de

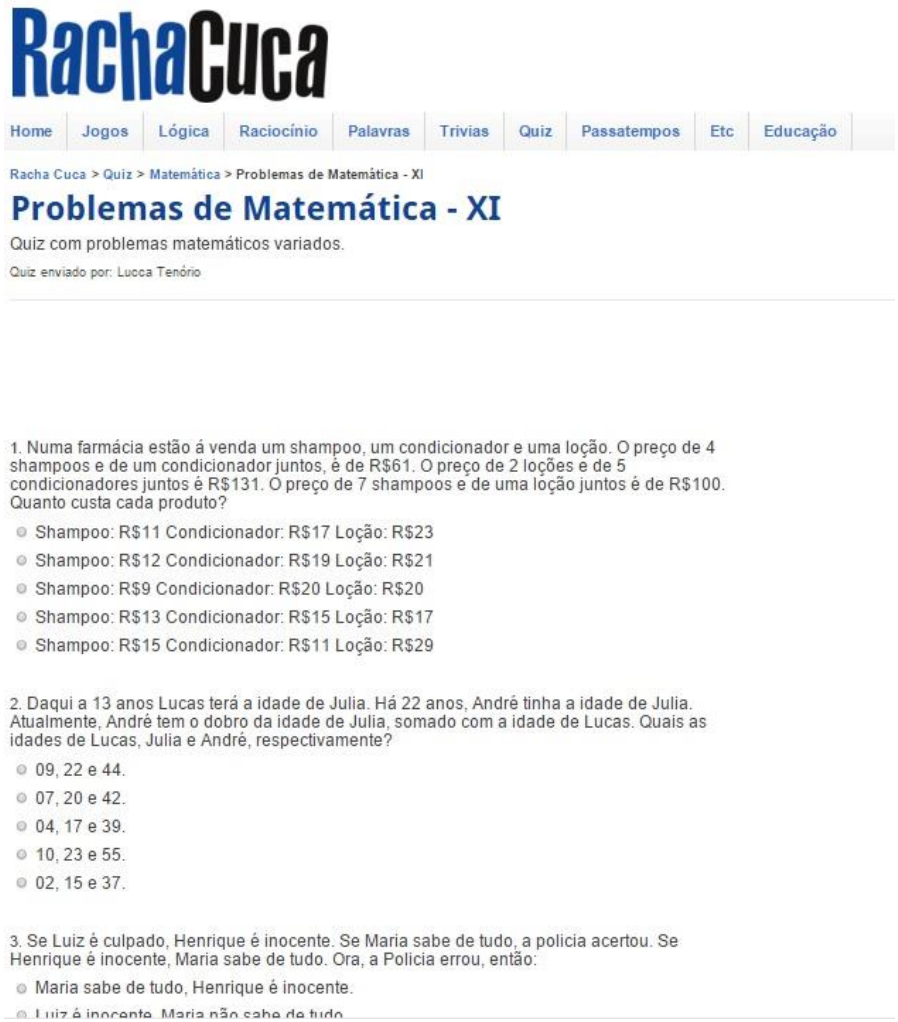
facilitar o aprendizado, sendo seu produto final um jogo digital que traz consigo conceitos de trigonometria, visando o estímulo de aprender, de forma menos pressionada, tal disciplina.

2.5 Exemplos de jogos e atividades digitais de Matemática

Esta seção abordará o funcionamento de jogos e atividades digitais que possuem como tema a disciplina de Matemática.

2.5.1 Problemas de Matemática

Figura 2.5 – Problemas de Matemática do site RachaCuca



Fonte: Captura do site RachaCuca[9].

[9] Disponível em: <http://rachacuca.com.br/quiz/matematica/>. Acesso em Abril de 2015

O site RachaCuca (2015) disponibiliza, entre outras atividades lúdicas, um *quiz* envolvendo temas de vários conteúdos dentro da área da Matemática, tais como equações do primeiro e segundo grau, regra de três, juros simples e probabilidade. O *quiz* possui uma divisão de níveis, onde o internauta pode escolher a dificuldade das perguntas. Ao decorrer da página o usuário responde várias preguntas sobre o tema e dificuldade que escolher e ao final da página, ele consegue conferir as repostas. O site mostra as respostas justificando-as, ajudando a tirar as dúvidas sobre as determinadas questões.

2.5.2 Matemática Financeira

Figura 2.6 – Aplicação de Matemática Financeira

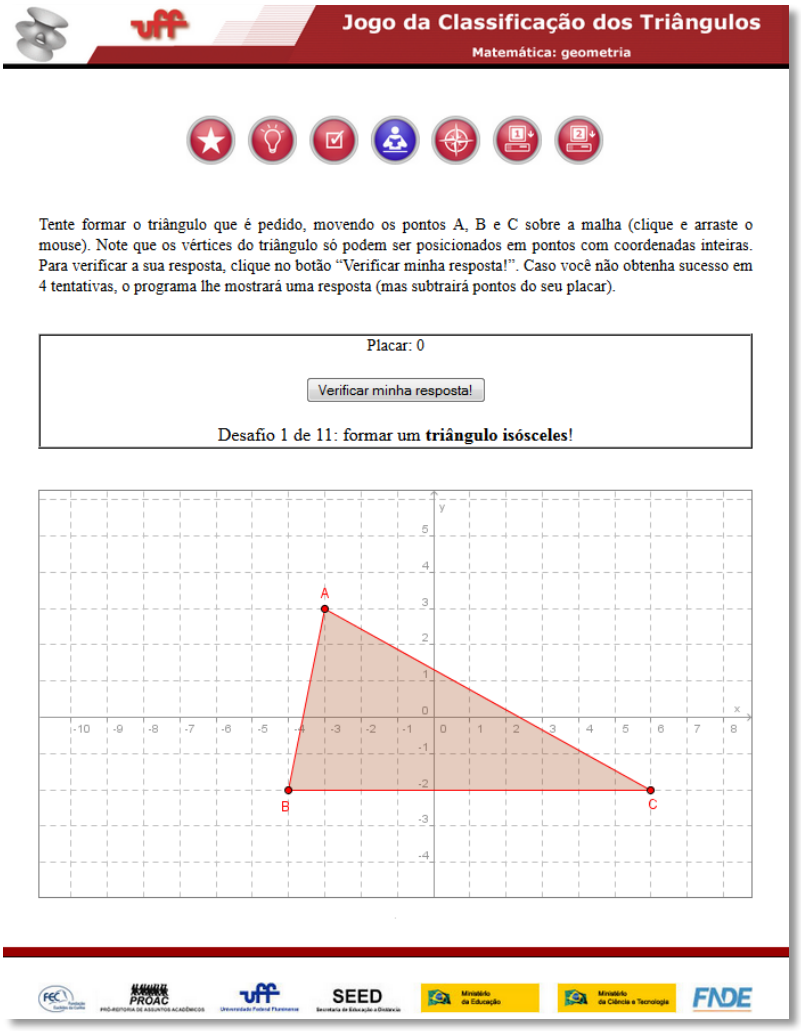
Fonte: Captura do site da UFF[10].

[10] Disponível em: <http://www.uff.br/cdme/poupanca/poupanca-html/poupanca_aplicacao-br.html>. Acesso em Abril de 2015

No site da UFF – Universidade Federal Fluminense (2015) possui diversos jogos envolvendo conteúdos diferentes da área da Matemática. Em uma das suas atividades o participante vê duas aplicações reais de Matemática Financeira, sendo a primeira sobre os rendimentos mensais da caderneta de poupança e a segunda, com o índice de inflação medida pelo INPC - Índice Nacional de Preços ao Consumidor. O internauta deverá responder a uma série de perguntas, ora fazendo cálculos na calculadora, ora arrastando valores para posições adequadas na tabela de resultados. Sendo possível também refazer as atividades com valores diferentes.

2.5.3 Jogo da Classificação dos Triângulos

Figura 2.7 – Jogo da Classificação dos Triângulos



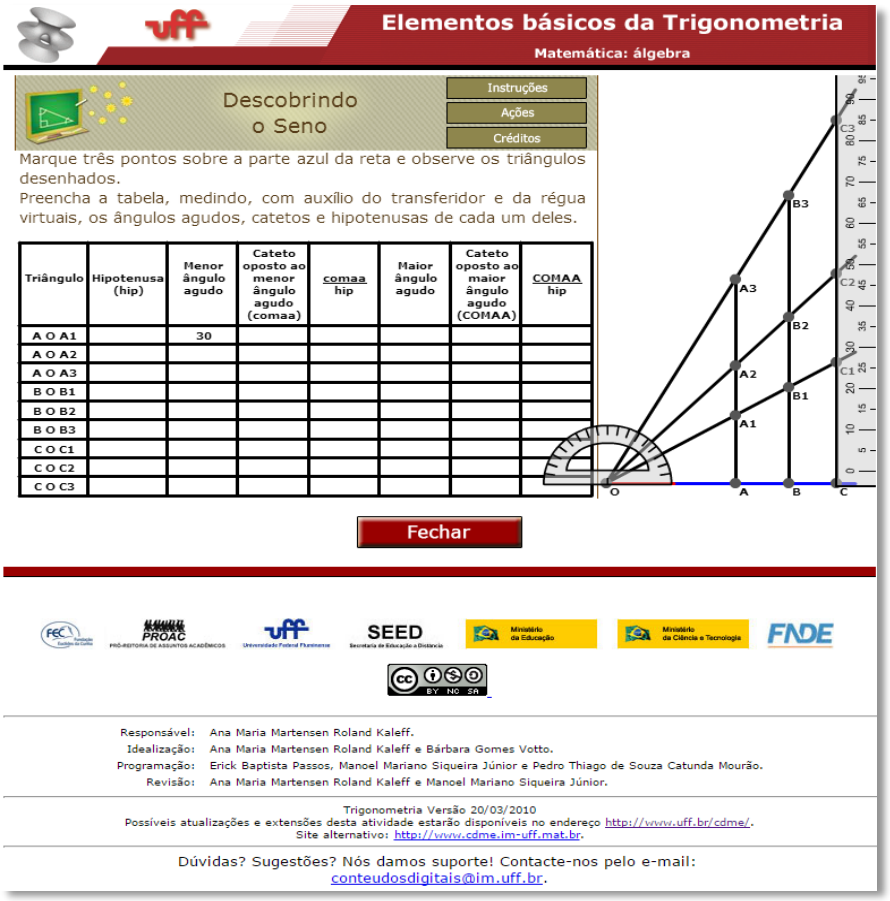
Fonte: Captura do site da UFF[11].

[11] Disponível em: <http://www.uff.br/cdme/jct/jct-html/jct-br.html>. Acesso em Abril de 2015

Outro jogo também encontrado no site da UFF (2015) é da Classificação dos Triângulos. O Jogo consiste em formar triângulos de acordo com a solicitação da atividade proposta. Para formar os triângulos deve-se mover os pontos indicado na malha como A, B e C, onde os vértices do triângulos só podem se posicionados em pontos com coordenadas inteiras. O jogador pode verificar a resposta e caso ela não consiga chegar no resultado desejado em quatro tentativas, o jogo lhe mostrará uma resposta subtraindo pontos do seu placar.

2.5.4 Descobrindo o Seno

Figura 2.8 – Jogo Descobrindo o Seno



Fonte: Captura do site da UFF[12].

Também no site da UFF (2015), pode-se encontrar uma atividade relacionada com elementos básicos da trigonometria. Chamada de Descobrindo o Seno, a atividade consiste em marcar três pontos em uma reta azul para se formar três

[12] Disponível em: <http://www.uff.br/cdme/trigonometria/jogo01/jogo01.html.>. Acesso em Abril de 2015

triângulos para serem observados e com o estudos destas formas geométricas deve-se preencher uma tabela os ângulos agudos, catetos e hipotenusa de cada uma delas. Para isso, o participante terá como ferramenta de ajuda uma régua e transferidor virtual. Em algumas células, quando o dado preenchido fica da cor verde quer dizer que foi feito de forma correta, e, caso fique vermelho, o preenchimento foi de forma indevida.

2.5.5 Math Mountain (Montanha Matemática)

Figura 2.9 – Jogo Math Mountain

Fonte: Captura do site UOL[13].

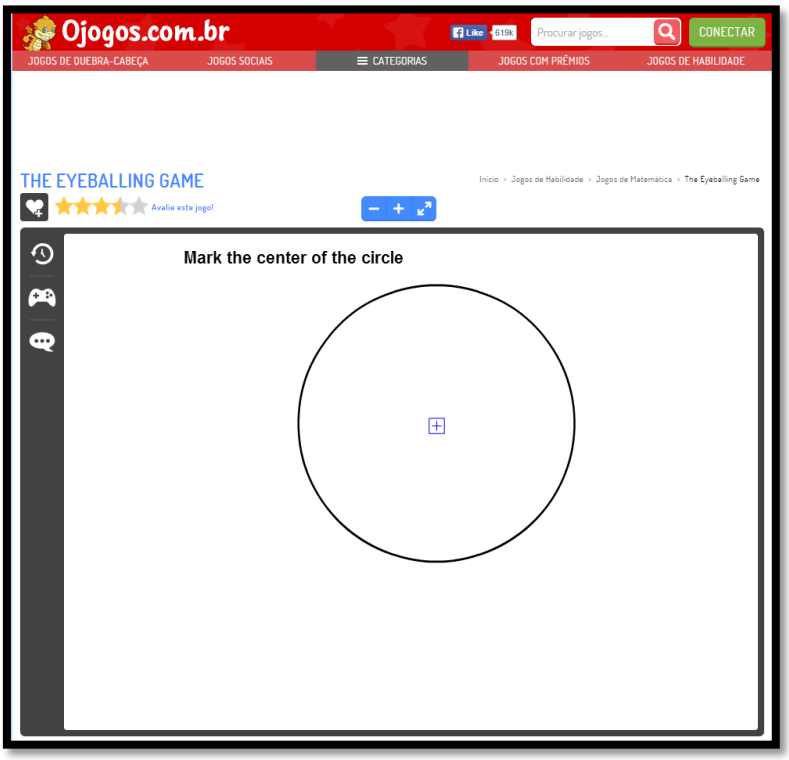
O jogo Math Mountain (Montanha Matemática), disponibilizado pelo site UOL (2015), trata-se de solucionar questões simples de Matemática. O personagem do jogador começa em um dos lados da base de uma montanha e seu adversário do lado oposto. Um questão é apresentada em uma janela com quatro alternativas e um tempo de dez segundos para responder. Caso o jogador responda a questão corretamente ele poderá subir um, duas ou três posições acima de acordo coma dificuldade da pergunta, mas caso ele responda de forma incorreta ele descerá o mesmo número de

[13] Disponível em: <http://jogos360.uol.com.br/math_mountain.html>. Acesso em Abril de 2015

posições. O objetivo do jogo é alcançar o topo da montanha antes do adversário.

2.5.6 The Eyeballing game (O jogo EyeBalling)

Figura 2.10 – Jogo Eyeballing



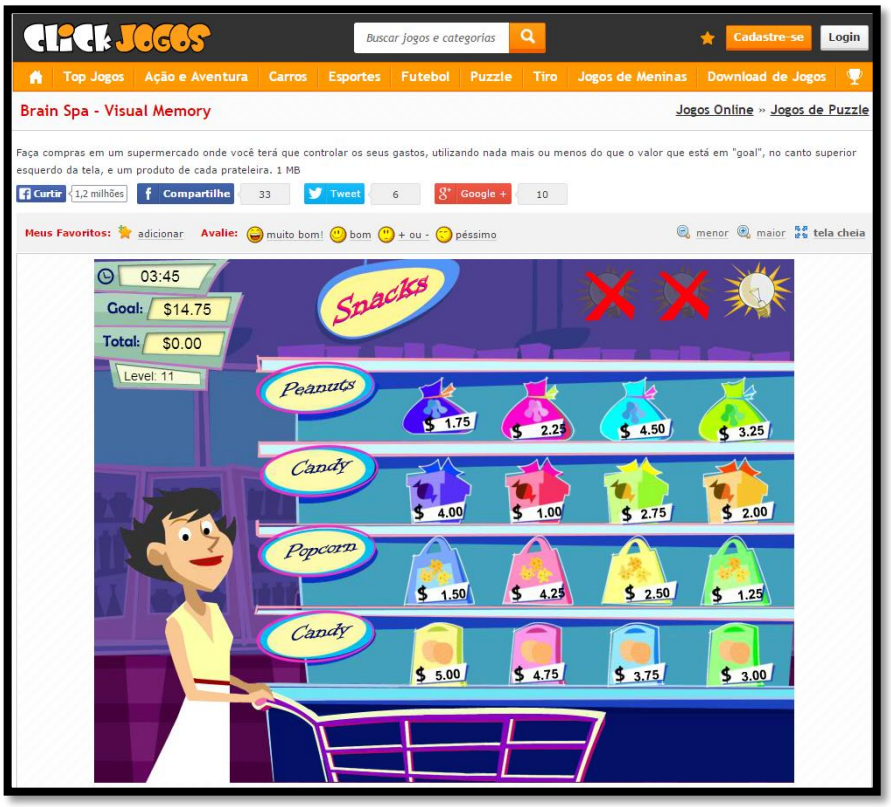
Fonte: Captura do site Ojogos.com.br[14].

O jogo Eyeballing pode ser encontrado no site Ojogos (2015). Neste jogo o participante deve arrastar um indicador em azul de forma a encontrar o que se pede. Esta atividade possui desafios como encontrar o centro de um círculo, achar o ângulo reto, encontrar o ponto de convergência, ajustar para fazer um paralelogramo, encontrar o ponto médio de um segmento, achar a bissetriz de um ângulo e marcar o pontos equidistante aos lados de um triângulo.

[14] Disponível em: <http://www.ojogos.com.br/jogo/the-eyeballing-game>. Acesso em Abril de 2015

2.5.7 Brain Spa (Spa Cérebro)

Figura 2.11 – Jogo Brain Spa



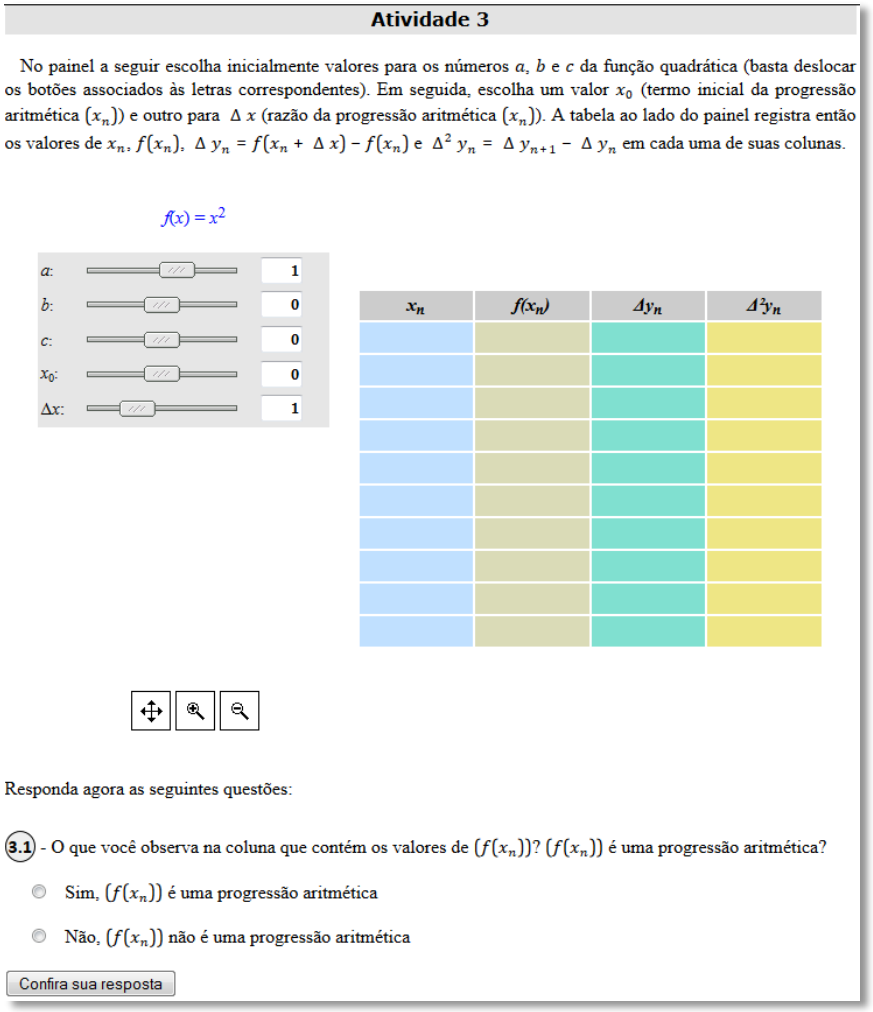
Fonte: Captura do site Click Jogos[15].

Brian Spa é um jogo disponibilizado pela Click Jogos (2015) que consiste simular compras em um spa, onde o jogador deve selecionar um item de cada prateleira de produtos de acordo que a sua soma tenha que dar o valor exigido pelo jogo. A medida que o usuário acerta o valor necessário, o jogo dificulta aumentando as prateleiras e as quantidades de produtos em cada uma delas. O jogador deve pensar na soma dos preços o mais rápido possível para alcançar o fim do desafio com o menor tempo possível.

[15] Disponível em: <http://www.clickjogos.com.br/Jogos-online/Puzzle/Brain-Spa-Visual-Memory/>. Acesso em Abril de 2015

2.5.8 Variação da função quadrática

Figura 2.12 – Atividade de função quadrática



Fonte: Captura do site da UFF[16].

Neste exercício disponibilizado pelo site da UFF (2015), o usuário deverá em um plano arrastar indicadores que correspondem e variam os valores dos pontos a, b e c, alterando assim o gráfico da função correspondente as posições desses pontos. A medida que o usuário faz suas escolhas, ele deve observar e conferir com a ideia que o site propõe e disponibiliza. O internauta pode fazer outras atividades relacionadas à medida que resolve, envolvendo valores de delta, progressão aritmética e definição de função quadrática.

[16] Disponível em: <http://www.uff.br/cdme/quadratica/quadratica-html/QP3.html>. Acesso em Abril de 2015.

2.5.9 Matemática Elementar

Figura 2.13 – Aplicativo Matemática Elementar

Fonte: Captura do dispositivo rodando o aplicativo Matemática Elementar.

O Matemática Elementar é um aplicativo disponibilizado pela Play Store[17] criado e desenvolvido pela Fábrica de Software[18] da UFMS (Universidade Federal do Mato Grosso do Sul) Campus de Ponta Porã. Através do aplicativo o usuário terá a oportunidade de obter informações básicas de alguns temas da matemática como, por exemplo, conjuntos numéricos, intervalos, e produtos notáveis. Ao final das informações de cada tema, é permitido ao usuário resolver atividades que envolva o assunto em questão.

[17] Loja de aplicativos do Android. Disponível em aplicativo pelo dispositivo ou pelo site da loja. Disponível em: <https://play.google.com/store?hl=pt_BR/>. Acesso em Abril de 2015.
[18] Disponível em: <www.fabricadesoftwarepp.com.br/>. Acesso em Abril de 2015.

2.5.10 Matemática: Treine seu Cérebro

Figura 2.14 – Aplicativo Matemática: Treine seu Cérebro

Fonte: Captura do dispositivo rodando o aplicativo em questão.

Esse jogo virtual é um aplicativo que também pode ser baixado pelo Play Store. O jogo conta com perguntas envolvendo operações (soma, subtração, multiplicação e divisão de números negativos e positivos) com dois e três argumentos. As perguntas são separadas em torno de 12 níveis aumentando a dificuldade ao decorrer da realização deles. O jogo pode ser utilizado de forma online com modo de multijogador, ou seja, podendo ser jogado com outras pessoas que também possuem o aplicativo. O aplicativo também disponibiliza o ranking dos melhores jogadores, sendo estes estipulados com a melhor pontuação dada a soma dos acertos obtidos em todos os níveis.

3 MÉTODOS E FERRAMENTAS UTILIZADAS

3.1 Método

O método de desenvolvimento de software utilizado para desenvolver a aplicação proposta foi o método ágil Scrum. Segundo Schwaber e Sutherland (2013), os pioneiros a aplicarem o método, com o Scrum os integrantes do projeto a ser desenvolvido conseguem lidar e solucionar problemas complexos de uma forma produtiva e criativa com o mais alto valor possível, sendo um método leve e simples de entender, organizando os processos em etapas, definindo as metas e permitindo respostas rápidas, e, portando, prevendo riscos. Essas características foram fundamentais para a adoção do método para desenvolver a aplicação proposta.

Ainda segundo esses autores:

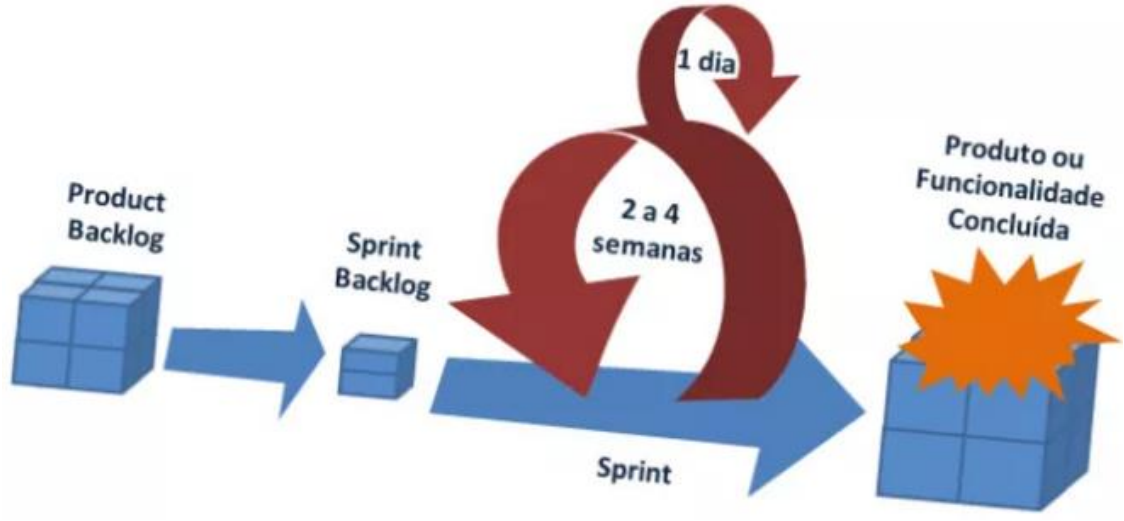
> Scrum é um *framework* estrutural que está sendo usado para gerenciar o desenvolvimento de produtos complexos desde o início de 1990. Scrum não é um processo ou uma técnica para construir produtos; em vez disso, é um *framework* dentro do qual você pode empregar vários processos ou técnicas. O Scrum deixa claro a eficácia relativa das práticas de gerenciamento e desenvolvimento de produtos, de modo que você possa melhorá-las (Schwaber e Sutherland, 2013, p. 3).

No Scrum, inicialmente, as funcionalidades e características do software a se desenvolver são estipuladas e mantidas em uma lista conhecida como Product Backlog. O projeto é dividido em ciclos/iterações comumente determinadas em um período de 1 a 4 semanas, chamados de Sprints.

Em cada Sprint, através de uma reunião chamada de Sprint Planning Meeting, os responsáveis pelo projeto, conhecidos como Product Owner prioriza os itens mais relevantes e a equipe determina um conjunto de atividades da Product Backlog que será capaz de realizar no período de uma Sprint, tal conjunto de atividades é determinado como Sprint Backlog. A final de uma Sprint a equipe mostra as funcionalidades e características que foram possíveis de se realizar neste ciclo e fazem uma retrospectiva de como foi a realização da Sprint, propondo melhorias e ações mais efetivas (SCHWABER e SUTHERLAND, 2013).

O ciclo da Scrum é representado pela Figura 3.1.

Figura 3.1 – Ciclo do Scrum



Fonte: Captura do site MindMaster[19].

Para este trabalho proposto, foram estipulados 4 Sprints com duração de 2 a 4 semanas, variando de acordo com a dificuldade exigida nas especificações de cada uma elas. Ao início de cada ciclo, ou seja, de cada Sprint, ocorreram as reuniões de planejamento (Sprint Planning Meeting), onde o responsável pelo projeto (*Product Owner*) priorizou todos requisitos a serem implementadas (*Product Backlog*). No caso deste trabalho, os envolvidos foram: aluno, orientador e o colaborador Alex Erickson.

As funcionalidades para serem desenvolvidas em cada Sprint, são selecionadas de acordo com que o desenvolvedor da aplicação será capaz de implementar durante a mesma que se inicia. Essas funcionalidades são transferidas para a lista de tarefas (Sprint Backlog).

As funcionalidades a serem desenvolvidas, selecionados para a primeira Sprint, que foi estipulada para o prazo de 2 semanas, foram cadastrar e deletar um novo usuário. A segunda Sprint, que teve o prazo de 3 semanas, teve como funcionalidades a serem desenvolvidas calcular e apresentar em uma tela específica o rendimento adquirido pelo usuário ao realizar o jogo. Para a terceira Sprint, com o prazo de 4 semanas, as funcionalidades indicar a exatidão obtida em cada resposta, apresentar uma imagem auxiliar ás perguntas e editar o usuário foram estipuladas para desenvolvimento. E por último, a quarta Sprint, com o prazo de 3 semanas, foi

[19] Disponível em: <http://www.mindmaster.com.br/scrum/>. Acesso em Janeiro de 2016

determinada para a implementação das funcionalidades disponibilizar e individualizar ajudas para as perguntas de acordo com o tema da mesma.

Ao andamento de cada ciclo ocorreram breves reuniões, conhecidas como *Daily Scrum*, de 10 a 15 minutos onde o aluno, desenvolvedor da aplicação, demonstrou o que já havia implementado, o que ainda falta desenvolver para atingir o que foi proposto para o ciclo em questão e se existe algo dificultando seu trabalho.

Além disso, ao final de cada Sprint, foram apresentadas versões beta o aplicativo, tendo o intuito de demonstrar o funcionamento do mesmo com as funcionalidades exigidas já implementadas e visando detectar o que foi realizado com exatidão, o que é preciso melhor, explorar novas ideias de funcionalidades que seriam importantes para o enriquecimento da aplicação e determinar as funcionalidades do próximo ciclo.

3.2 Sistema Operacional

O *Show da Trigonometria* foi desenvolvido para o sistema operacional Andorid. Com o intuito de ser uma ferramenta útil e acessível para o maior números de alunos possíveis, tal decisão foi tomada mediante a pesquisas que informam que esse sistema é o mais utilizado no mundo. A escolha também foi feita por ser um sistema Open Source, ou seja, um sistema aberto.

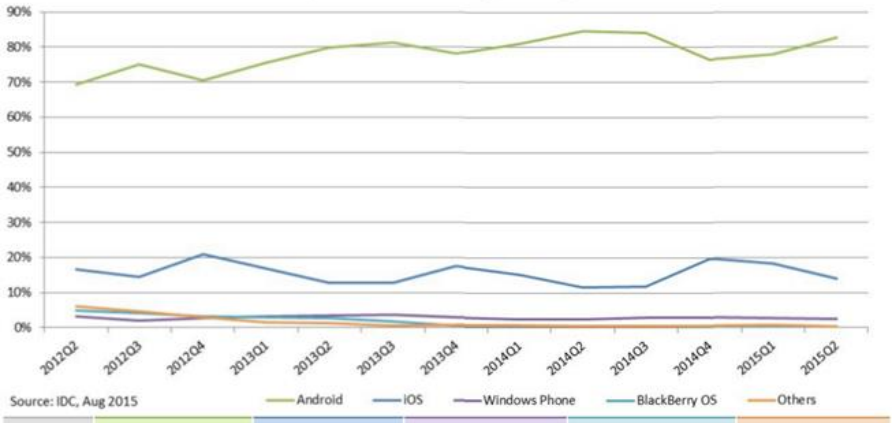
De acordo com a International Data Corporation (IDC), o Andorid dominou o mercado com uma cota de 82,8% dos 341,5 milhões de smartphones vendidos, no segundo semestre de 2015. Em segundo lugar ficou o iOs, sistema operacional da Apple, seguido do Windows Phone da Microsoft com 2,6% e o BlackBerry com 0,3%. Como mostra o Gráfico 3.1, o Android vem dominando o mercado e sua utilização vem crescendo desde o segundo semestre de 2012.

Android é um sistema operacional da Google, baseado em um sistema operacional Linux, com foco para smartphones e tablets. É um sistema Open Source, ou seja, qualquer um pode usar, modificar e distribuir o sistema, facilitando assim a vida dos desenvolvedores. Está presente também em câmeras digitais, Smart Tv’s e consoles de videogame (LECHETA, 2010).

O Android foi desenvolvido por empresários que já tinham experiência na área de tecnologia, estes foram Andy Rubin, Rich Miner, Nick Sears e Chris White, os mesmo fundaram a Android Inc. e optavam por fazerem de forma oculta os desenvolvimento de seus projetos. O sistema operacional surgiu em 2003, na cidade

de Palo Alto na Califórnia (SUPER INTERESSANTE, 2016).

Gráfico 3.1 – Quota global de mercado detida pelos principais sistemas operacionais de smartphones nas vendas para usuários do primeiro trimestre de 2012 até o segundo trimestre de 2015



Period	Android	iOS	Windows Phone	BlackBerry OS	Others
2015Q2	82.8%	13.9%	2.6%	0.3%	0.4%
2014Q2	84.8%	11.6%	2.5%	0.5%	0.7%
2013Q2	79.8%	12.9%	3.4%	2.8%	1.2%
2012Q2	69.3%	16.6%	3.1%	4.9%	6.1%

Fonte: Captura do site do IDC[20].

Em 2005, a Android Inc. foi comprada pela Google, sendo uma incógnita para muitos, pois o mercado já era consolidado pelo o Windows Phone, da Microsoft e liderado pelo iOS da Apple, mas a Google viu como uma grande oportunidade para se alavancar neste tipo de segmentação do mercado. Em 2007 a Google e grandes empresas como Sony, HTC, Sprint Nextel e Qualcomm se reuniram e fundaram a Open Handset Alliance, criando o primeiro o primeiro Android comercial, “Android 1.0” rodando no HTC G1 Dream, o qual pode ser visto na Figura 3.2, lançado em 22 de outubro de 2008.

Desde então o Android veio crescendo e hoje está presente nos aparelhos mais desejados do mercado, como o G Flex 2 da LG e o Galaxy S6 da Samsung, se tornado

[20] Disponível em: <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>. Acesso em Dezembro de 2015

o sistema mais utilizado do mundo. Atualmente as versões do Android são desenvolvidas sob o codinome de um doce e lançadas em ordem alfabética: Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop e a versão mais atual 6.0 Marshmallow, lançada oficialmente no dia 29 de setembro de 2015 (OFICIAL DA NET, 2016).

Figura 3.2 – HTC G1 Dream

Fonte: Captura do site PHANDROID[21].

3.3 Linguagens Utilizadas

3.3.1 Java

Uma das linguagens utilizadas para o desenvolvimento do *Show da Trigonometria* é a linguagem Java. Segundo Flanagan (2006), com o Java é possível escrever a aplicação somente uma vez para que possa ser executadas em qualquer lugar que suporte a plataforma Java, como os navegadores WEB. Também, segundo o autor, a linguagem foi projetada tendo em vista a segurança e a totalidade dos idiomas. É uma linguagem dinâmica e extensível sendo orientada a objetos e a linguagem é elegante combinada com um conjunto de APIs (Application Programming Interface, traduzido como Interface de Programação de Aplicações) poderoso e bem

[21] Disponível em: < http://phandroid.com/htc-dream/>. Acesso em Janeiro de 2016

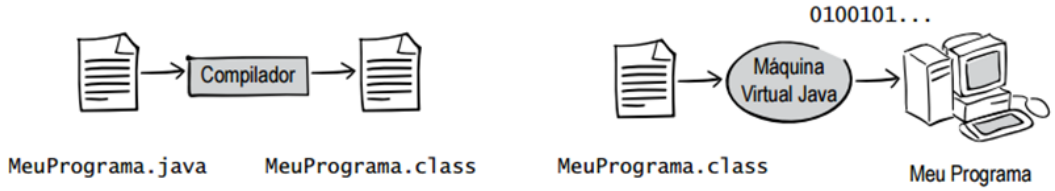
projetado. Essas características foram pontos importantes pela a escolha da linguagem.

A linguagem Java foi desenvolvida na década de 90 por uma equipe da Sun Microsystem liderada por James Gosling. Seu lançamento teve como foco os clientes browsers, ou seja, desenvolvedores de aplicações para internet, mas depois expandiram a linguagem para aplicações desktop e mobile. O objetivo dos seus desenvolvedores era criar uma linguagem mais simples, orientada a objetos e que fosse possível aprender de uma forma mais fácil, não ficando só de conhecimento aos mais experientes na programação (JUNIOR, 2002).

As classes contendo o código Java nessa aplicação nada mais é que um conjunto de instruções que descrevem uma tarefa a ser realizada pelo computador que dão o funcionamento para a aplicação. Segundo Junior (2002) o computador não entende diretamente essas instruções, para isso, quando compilado, o código gera o bytecode, que é uma forma intermediaria de código que será interpretado por uma máquina virtual, a Máquina Virtual Java (JVM). “[..] a máquina virtual traduz as instruções do código Java para instruções válidas no sistema operacional em que está rodando” (Semana da Tecnologia da Informação, 2006).

A Figura 3.3 demonstra o processo para interpretação do código Java.

Figura 3.3 – Processo para interpretação do código Java



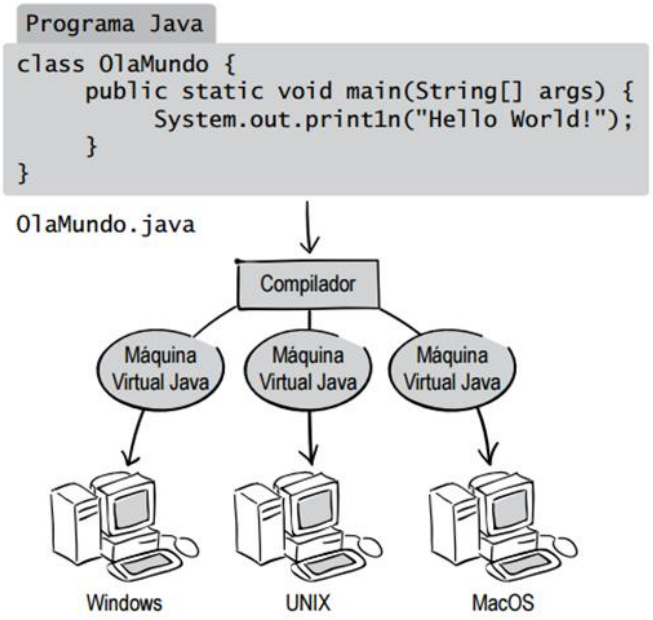
Fonte: (MENDES, 2009).

Uma das grandes vantagens de se desenvolver em Java é que a JMV pode estar presente em várias plataformas, sendo assim um programa na linguagem Java pode rodar em diferentes sistemas operacionais, como Windows, Linux ou Mac OS, fazendo necessário a presença da JMV (FLANAGAN, 2006). A Figura 3.4 representa um mesmo programa em Java sendo possível ser executado em múltiplos sistemas operacionais.

Logo, nos arquivos de extensão Java do *Show da Trigonometria* estarão

presentes a parte lógica da aplicação, ou seja, as instruções necessárias para a aplicação rodar.

Figura 3.4 – Programa Java é executado em múltiplas plataformas



Fonte: (MENDES, 2009).

3.3.2 XML

Outra linguagem utilizado no desenvolvimento do *Show da Trigonometria* é a XML, do inglês eXtended Markup Language, é uma linguagem de marcação, um padrão de formatação de dados, sendo uma forma organizada de guardar informações. Tal organização das informações é feita através de tags (DEITEL, 2003). A Figura 3.5 demonstra a formação de um código em XML. Segundo Zanella (2011, p. 11) utilizando XML “é muito mais fácil identificar e trabalhar com as informações, pois elas estão bem estruturadas e separadas por tags”.

No desenvolvimento para Android os arquivos XML são direcionados a representação gráficas das aplicações para instanciar botões e outros componentes

visuais presentes nas telas do aplicativo, juntamente com as especificações de tamanho, cor entre outras características (LECHETA, 2010).

Figura 3.5 – Exemplo da formação de um código em XML

```
<usuarios>
   <usuario id="0">
      <nome>
         <primeiro>Fulano</primeiro>
         <sobrenome>De Tal </sobrenome>
      </nome>
      <email>fulanotal@gmail.com</email>
   </usuario>
   <usuario id="1">
      <nome>
         <primeiro>Beltrano </primeiro>
         <sobrenome>De Tal </sobrenome>
      </nome>
      <email>beltranotal@gmail.com</email>
   </usuario>
</usuarios>
```

Fonte: (ZANELLA, 2011).

No *Show da Trigonometria* os arquivos XML foram usados para se ter a separação da parte lógica da parte gráfica. A parte logica estará nos arquivos de extensão Java e a parte gráfica nos arquivos XML. Segundo Lecheta (2010), a separação dos arquivos XML do código-fonte utilizando a API Java é feita para deixar o código mais fácil de se entender e com maior facilidade de manutenção.

3.4 Ferramentas utilizadas

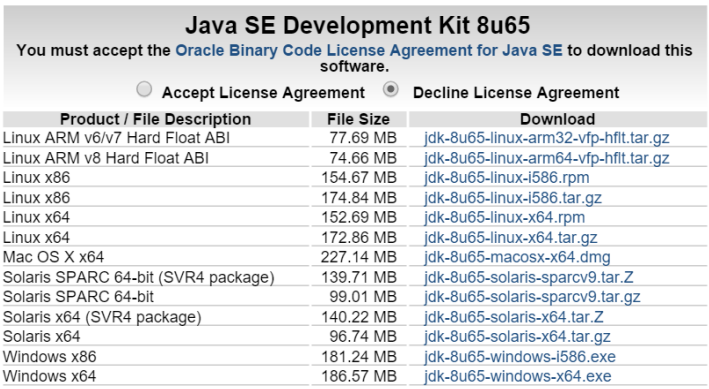
3.4.1 Java Development Kit (JDK)

Para a criação da aplicação *Show da Trigonometria* foi necessário, primeiramente, instalar o Java Development Kit (JDK). Segundo o site da Oracle (2016) o JDK é um pacote completo de desenvolvimento que inclui a máquina virtual

Java juntamente com o compilador e as bibliotecas necessárias para programar qualquer tipo de aplicação. Faz-se importante instalar o JDK de versão mais atual para usufruir mais de suas ferramentas.

O JDK está disponível de forma gratuita no site da empresa Oracle, bastando apenas acessar o link www.oracle.com ir na seção de downloads, depois em “Java para Desenvolvedores”, ler e aceitar o termo de licença e escolher a versão e o download que se enquadra nas configurações do computador a se instalar o JDK. A Figura 3.6 mostra os diferentes tipos de downloads possíveis para diferentes plataformas.

Figura 3.6 – Downloads possíveis para o JDK



Java SE Development Kit 8u65

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ Accept License Agreement ◉ Decline License Agreement

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	77.69 MB	jdk-8u65-linux-arm32-vfp-hflt.tar.gz
Linux ARM v8 Hard Float ABI	74.66 MB	jdk-8u65-linux-arm64-vfp-hflt.tar.gz
Linux x86	154.67 MB	jdk-8u65-linux-i586.rpm
Linux x86	174.84 MB	jdk-8u65-linux-i586.tar.gz
Linux x64	152.69 MB	jdk-8u65-linux-x64.rpm
Linux x64	172.86 MB	jdk-8u65-linux-x64.tar.gz
Mac OS X x64	227.14 MB	jdk-8u65-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.71 MB	jdk-8u65-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.01 MB	jdk-8u65-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.22 MB	jdk-8u65-solaris-x64.tar.Z
Solaris x64	96.74 MB	jdk-8u65-solaris-x64.tar.gz
Windows x86	181.24 MB	jdk-8u65-windows-i586.exe
Windows x64	186.57 MB	jdk-8u65-windows-x64.exe

Fonte: Captura do site da Oracle[22].

3.4.2 Android Studio

Para o desenvolvimento do trabalho proposto foi utilizado o Ambiente de Desenvolvimento Integrado (do inglês *Integrated Development Enviroment* – IDE) conhecido como Android Studio.

Segundo Sebesta (2000, p. 46, citado por CHAVES e SILVA, 2008, p.2), IDE pode ser identificado como um ambiente de desenvolvimento integrado que reúne características e ferramentas que dão apoio ao desenvolvimento de software, com o objetivo de agilizar o processo.

A escolha pelo Android Studio se deu ao fato de o mesmo, de acordo com o

[22] Disponível em: <http://www.oracle.com/>. Acesso em Janeiro de 2016

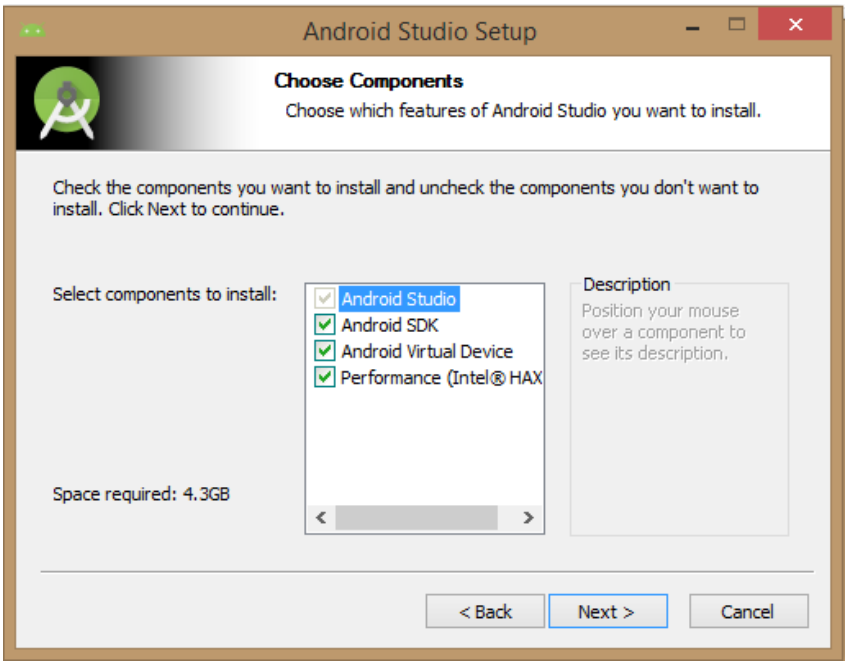
site do Andorid (2016), ser o IDE oficial do sistema operacional Android.

O Android Studio possui assistentes de projetos que ajudam a criar e iniciar um novo projeto com modelos padrões, editor de *layout* enriquecido com auxílio para edição de arrastar e soltar temas desejado, entre outras ferramentas úteis e atualizadas (ANDROID, 2016).

3.4.2.1 Instalação

Para instalar o Android Studio é necessário fazer o seu download no site da empresa Android no link http://developer.android.com/intl/pt-br/sdk/index.html. No guia de instalação terá a opção da instalação das versões mais atualizadas do kit do desenvolvimento SDK (*Software Development Kit*) do Android e do AVD (Android Virtual Device), como pode ser visto na Figura 3.7.

Figura 3.7 – Guia de instalação do Android Studio

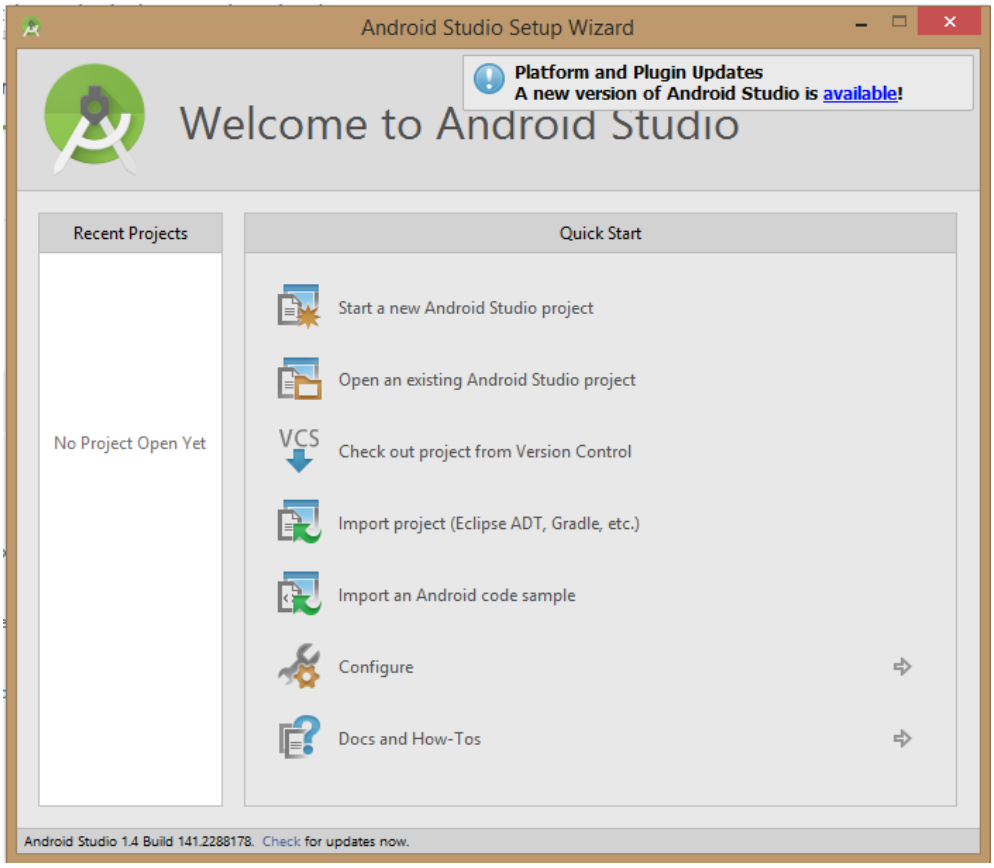


- **SDK:** Segundo Lacheta (2010, p. 30) o Android SDK é um software para desenvolvimento de aplicações no Android que possui ferramentas utilitárias e uma interface de programação de aplicativos completa para a linguagem Java, tendo todas as classes necessárias para o desenvolvimento das aplicações.

- **AVD:** É um emulador criado para “simular exatamente um configuração de um celular real, com exatamente a mesma plataforma do sistema operacional, resolução de tela e outras configurações” (LACHETA, 2010, p. 38).

Após escolher as devidas configurações de preferência e concluir o guia de instalação, abrirá uma janela oferecendo opções como criar um novo projeto, abrir um projeto já existente e configurar o ambiente de desenvolvimento instalado, como pode ser visto na Figura 3.8.

Figura 3.8 – Janela após a instalação do Android Studio

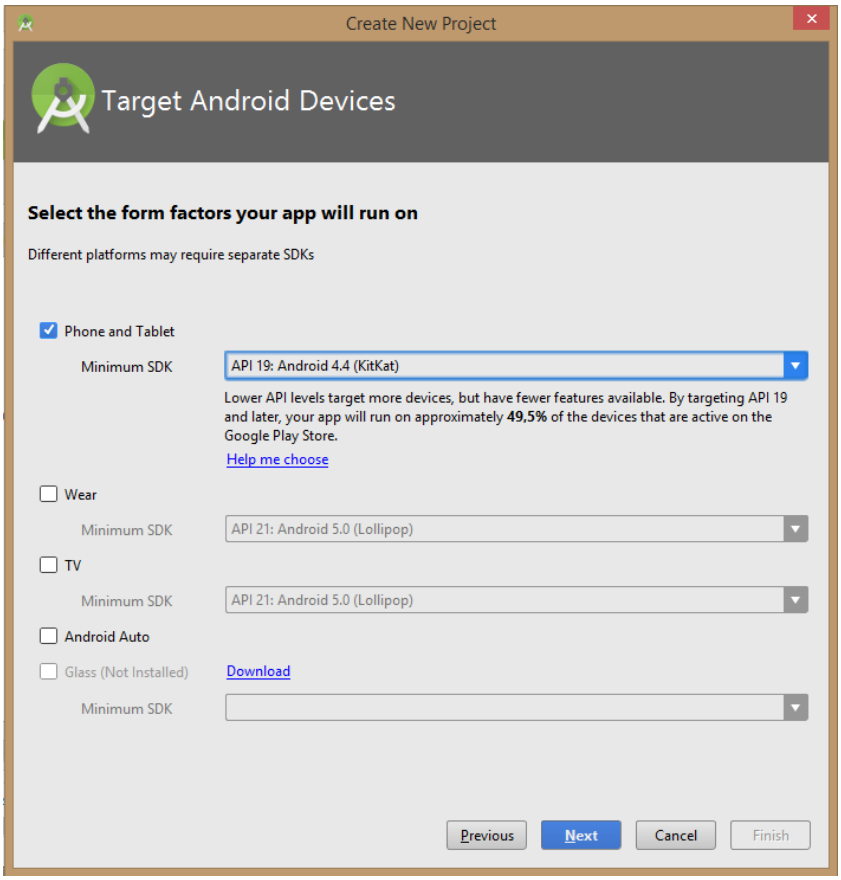


3.4.2.2 Novo projeto no Android Studio

Para criar um novo projeto no Android Studio basta acessar a barra de ferramentas e escolher a opção *Novo Projeto*. Em seguida, abrirá uma janela para ser informado o nome do projeto e o diretório que ele deve ser salvo. Após isso, deverá

escolher a versão do Android a qual vai ser direcionada a aplicação a ser criada, como pode ser visto pela Figura 3.9.

Figura 3.9 – Escolhendo a versão do Android



A versão KitKat é a mais utilizada entre as demais versões do Android, conforme Tabela 3.1. Assim sendo, esta foi escolhida para desenvolvimento do aplicativo proposto neste trabalho.

Após escolher a versão será necessário escolher configurações para a *activity* inicial do projeto como o nome e o tema, como é ilustrado pela figura 3.10. A definição de *activity* se encontra na seção 3.4.2.4.

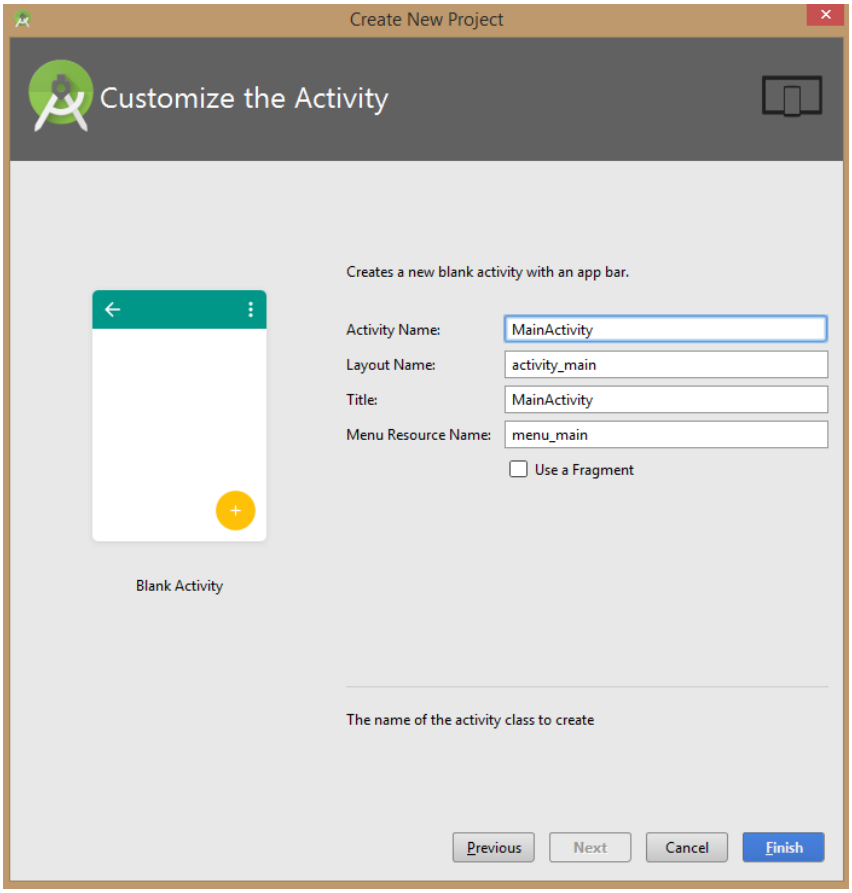
Tabela 3.1 – Versões do Android mais utilizadas

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.7%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.5%
4.1.x	Jelly Bean	16	8.8%
4.2.x		17	11.7%
4.3		18	3.4%
4.4	KitKat	19	35.5%
5.0	Lollipop	21	17.0%
5.1		22	17.1%
6.0	Marshmallow	23	1.2%

Fonte: Captura do site do Android[23].

Figura 3.10 – Configurando a *activity* inicial

[23] Disponível em: <http://developer.android.com/intl/pt-br/about/dashboards/index.html>. Acesso em Fevereiro de 2016



Após a conclusão da configuração da *activity*, abrirá o ambiente de desenvolvimento com procedimentos e configuração pré-estabelecidas para nortear a implementação. Agora bastar desenvolver de acordo com a aplicação que se deseja.

3.4.2.3 Activity

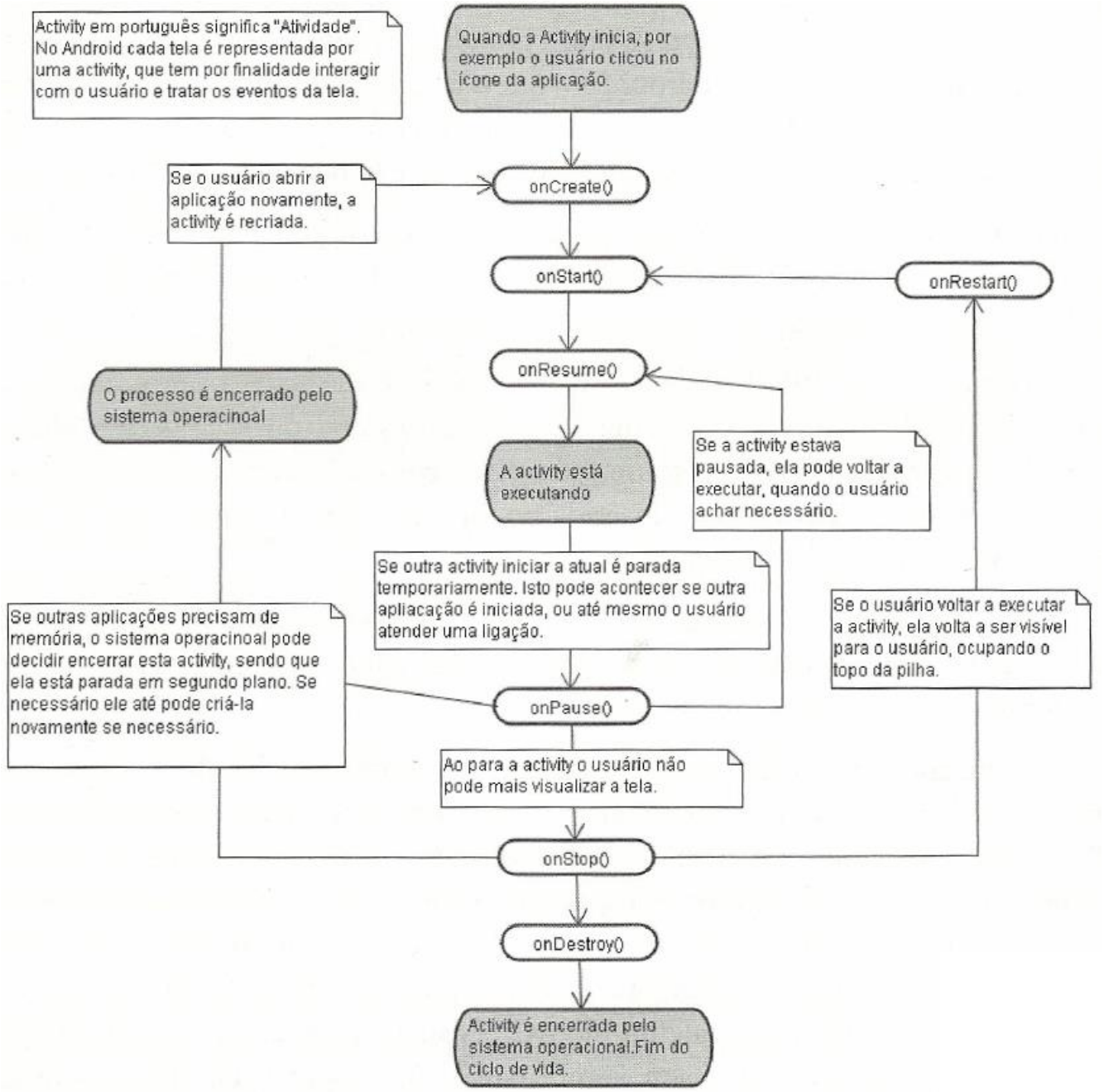
A a*ctivity* representa uma tela da aplicação, traduzindo para o português teríamos atividade, ou seja, são as ações que serão processadas quando o utilizador da aplicação interagir com os componentes visuais presentes na tela que ele está usufruindo. Pertencente a classe *andorid.aap.Activity*, uma *activity* é uma classe gerenciadora de interface com usuário (LANCHETA, 2010).

Como explanado por Lancheta (2010) uma *activity* possui seu ciclo de vida, que vai desde sua criação até seu encerramento. O diagrama mostrado na Figura 3.11, representa um ciclo de vida completo. É importante o uso dos métodos nas mudanças de estado de uma *activity*, permitindo ao desenvolvedor manipular sua aplicação.

Para melhor entendimento do ciclo de vida, é importante entender seus métodos que serão explanados a seguir.

- **onCreate():** Esse método é obrigatório é será chamado apenas uma vez. Será responsável por carregar o *layout*, podendo adicionar procedimentos que deverão ser executados antes da aparição visual da *activity*, ou seja, da tela;
- **OnStart():** É chamado posteriormente ao método *onCreate()* ou quando uma *activity* que estava em segundo plano é novamente chamada, ou seja, quando a tela está fincando visível ao usuário;
- **onResume():** Será chamado após o método *onStart()*, quando a *activity* está pronta para interagir com o usuário;
- **onPause():** Esse método será chamado quando a aplicação é interrompida, salvando o estado da aplicação no ato de interrupção;
- **onStop():** Esse método será chamado quando outra *activity* é chamada;
- **onRestart():** Sua chamada se faz quando uma *activity* está retomando do segundo plano. Este método chama automaticamente o método *OnStart()*;
- **onDestroy():** Ocorrerá logo antes da *activity* ser finalizada. Pode ser chamado pelo sistema operacional quando julgar necessária a liberação de recursos ou pela aplicação utilizando o método *finish()*.

Figura 3.11 – Ciclo de vida de uma *Activity*



Fonte: Livro Google Android[24].

3.4.2.4 Criando e configurando o AVD

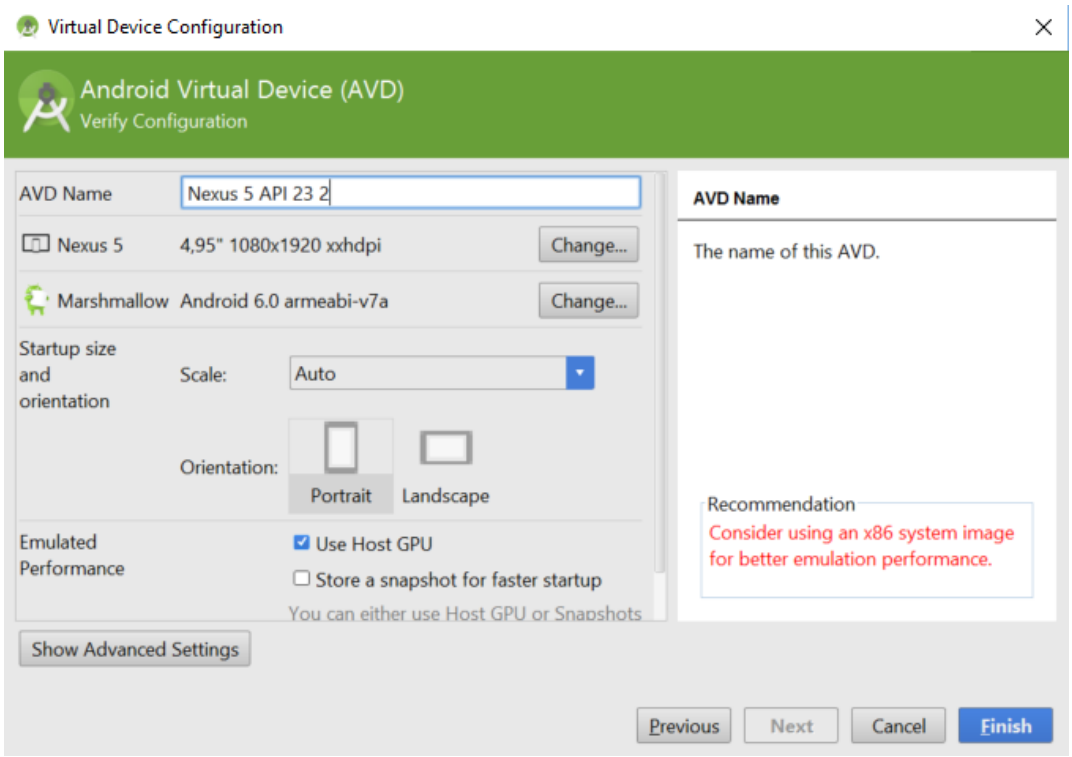
Para criar um AVD basta ir na aba Tools > Android > AVD Manager. Após a janela, demonstrada pela Figura 3.12, se abrir basta clicar em *Criar Virtual Device*.

[24] LACHETA, Ricardo R. **Google Android:** aprenda a criar aplicações para dispositivos móveis com Android SDK. 2 ed. São Paulo: Novatec Editora, 2010.

Feito isso basta escolher algumas configurações como novo do AVD, o dispositivo e a versão que ele vai emular, como demonstra a Figura 3.13.

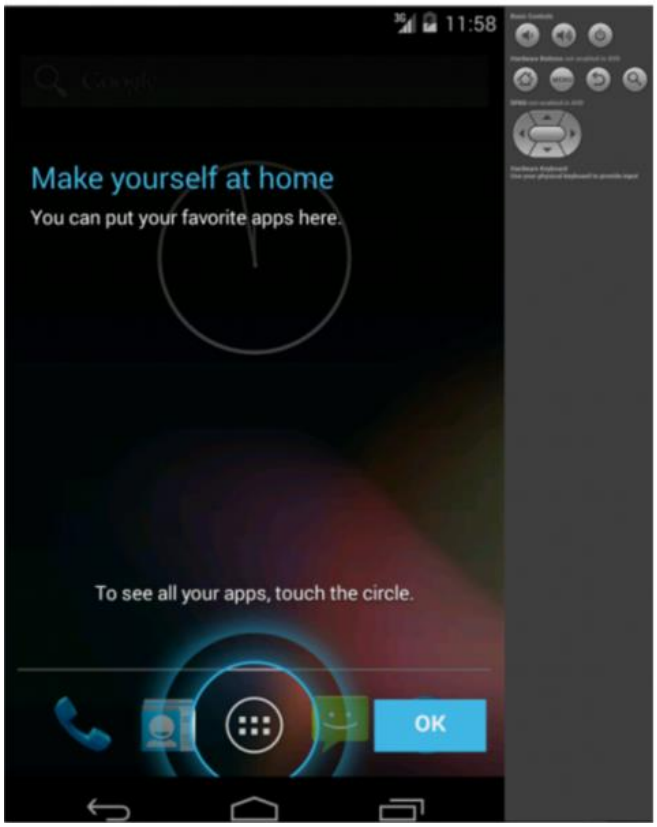
Figura 3.12 – Criando um novo AVD

Figura 3.13 – Configurando o AVD



Quando rodar a aplicação no ato do desenvolvimento poderá se optar por emulá-la no AVD criado. A Figura 3.14 exemplifica o emulador em funcionamento.

Figura 3.14 – Emulador Android em Funcionamento



4 DESCRIÇÃO DA APLICAÇÃO

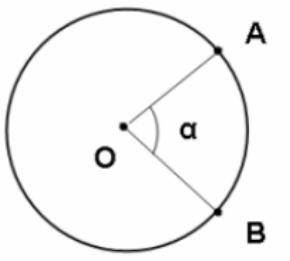
4.1 Definição

A aplicação proposta será conhecida como *Show da Trigonometria*, que é um jogo estilo *quiz*. *Quiz*, que é definido pelo site GeoEscola (2016) como “uma forma de avaliar uma grande quantidade de pessoas com um questionário com respostas curtas e de escolha múltipla visando a avaliação/auto avaliação”. O *Show da Trigonometria* é uma ferramenta lúdica que testará os conhecimentos de seus usuários na área de trigonometria, de forma a incentivá-los a buscar uma maior pontuação no jogo, ou seja, uma forma divertida que instiga a adquirir conhecimento.

As questões contidas no jogo são baseadas e classificadas em três temas abordados na área de Trigonometria, esses assuntos são:

- **Ângulos e Arcos:** De acordo com Iezze (1995, p. 1-C), “dados dois pontos distintos A e B sobre uma circunferência, esta fica dividida em duas partes. Cada uma dessas partes, que incluem A e B, é denominada arco de circunferência AB”, representado pela Figura 4.1. Segundo o site RIVED (2016) “a cada arco tomado corresponde um *ângulo central* e a medida de um arco equivale à medida do ângulo central correspondente.”

Figura 4.1 – Dois pontos sobre uma circunferência



Fonte: Captura do site Matemática[25].

- **Funções e Relações Trigonométricas:** Funções trigonométricas são funções angulares, importantes no estudo dos triângulos e na modelagem de fenômenos periódicos. As relações entre os valores das

[25] Disponível em: <http://www.uff.br/cdme/poupanca/poupanca-html/poupanca_aplicacao-br.html>. Acesso em Janeiro de 2016

funções trigonométricas de um mesmo arco são denominadas relações trigonométricas (EBAH, 2016).

- **Triângulos:** De acordo com o site eCalculo (2016), “um triângulo é uma figura geométrica plana, constituída por três lados e três ângulos internos. Esses ângulos, tradicionalmente, são medidos numa unidade de medida, denominada grau e, cada um deles tem medida entre 0º e 180º, de modo que, em qualquer triângulo, a soma dessas medidas é 180º”. Nessa seção o *quiz* utilizará de perguntas sobre as relações envolvendo os lados e os ângulos dos triângulos retângulos e de triângulos quaisquer.

Cada um desses temas apresentados é um módulo do jogo. Logo, são três módulos e cada módulo possui um grupo de perguntas associadas a eles de acordo com os assuntos que representam. A cada pergunta, será apresentado 4 respostas possíveis, sendo que somente uma é a correta para resolução do problema. Optando e confirmando sua escolha o jogador receberá uma mensagem afirmando se a sua escolha é a reposta exata. Certas perguntas da aplicação fazem necessário uso de imagens para complementá-las. Assim, as imagens são apresentadas juntamente a pergunta.

Para jogar o *Show Da trigonometria*, o utilizador do jogo faz um breve cadastro, informando apenas seu nome de usuário, ou seja, um nome de como que gostaria de ser abordado pela aplicação ao realizar tal atividade. Após o cadastro, tem-se a possibilidade de alterar e/ou deletar o nome de usuário cadastrado. O utilizador da aplicação poderá realizar quantos cadastros julgar necessário.

Para a realização do *quiz*, é necessário escolher entre um dos usuário cadastrados, depois de tal escolha será necessário a cada participação na aplicação, optar por um dos dois modo de se jogar: Modo Campanha e Modo Aleatório. O Modo Campanha é aonde o usuário resolverá o *quiz* em seções, tais seções são os 3 módulos já definidos. Para cada módulo, serão apresentadas 5 perguntas, sendo cada módulo respondido por vez até completar os 3 módulos, respondendo ao todo 15 perguntas. Ao final de cada módulo e do modo de jogo, a aplicação apresenta um mensagem da percentagem dos acertos de acordo com as perguntas respondidas.

O Modo Aleatório apresenta uma abordagem diferente do Modo Campanha.

No intuito de que o usuário consiga responder qualquer pergunta que o *quiz* apresentar, independente do módulo que ela pertença, este modo de jogo apresenta 15 perguntas em sequência independente do seu módulo, ou seja, 15 perguntas aleatórias entre todas que a aplicação oferece, para que o usuário possa responder. No final deste modo de jogo, ou seja, respondidas as 15 perguntas aleatórias, a aplicação mostra uma mensagem informando a percentagem de acertos obtidos. As percentagens correspondem a pontuação adquirida pelo jogador.

Caso o utilizador da aplicação desejar verificar a pontuação para um determinado usuário cadastrado, há uma tela com as pontuações adquiridas em cada módulo do Modo Campanha e a pontuação total dos dois modos de jogos oferecidos. De acordo com o professor atual da disciplina Fundamento de Matemática da UFVJM, as pontuações serão um ponto forte do aplicativo para que o utilizador da aplicação veja em qual módulo está com mais dificuldade, podendo incentivá-lo a aprofundar os estudos nessa seção a qual apresenta dificuldades e a buscar ajuda do professor da disciplina Fundamentos de Matemática atual.

Se o participante do jogo escolher por sair do jogo sem realizar por completo um modo de jogo, ele terá a opção de continuar a jogar posteriormente com o aproveitamento da sua última participação ou começar um novo jogo, zerando suas pontuações adquiridas.

4.2 Levantamento de Requisitos

Para a realização do levantamento dos requisitos que a aplicação deveria atender utilizou-se a técnica Brainstorming (tempestade de ideias).

Segundo Pinto (2007, p.2):

> A técnica Brainstorming é muito utilizada para promover a interação de um pequeno grupo de trabalho, onde o ponto chave está no incentivo à participação de todos em divulgar as ideias que vão surgindo. Assim, um grupo se reúne com a finalidade de obter o maior número possível de idéias para a solução de um problema específico.

Foram realizadas reuniões envolvendo o professor atual da disciplina de Fundamentos de Matemática e a orientadora Luciana Pereira de Assis, com intuito de se obter e discutir novas ideias para o desenvolvimento da aplicação.

Durante o processo de desenvolvimento, foram mostradas aos envolvidos, versões em andamento da aplicação visando a validação dos requisitos propostos.

4.2.1 Requisitos Essenciais

- Cadastrar um novo usuário;
- Deletar um usuário;
- Calcular o aproveitamento conquistado;
- Indicar a exatidão obtida em cada resposta.

4.2.2 Requisitos Desejáveis

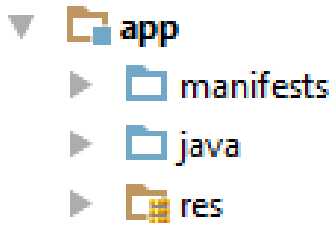
- Demonstrar uma imagem auxiliar para a pergunta;
- Disponibilizar uma ajuda para as perguntas;
- Individualizar as ajudas de cada módulo;
- Editar um usuário.

Após o levantamentos dos requisitos, deu-se início ao desenvolvimento do aplicativo. As Seções seguintes apresentam detalhes técnicos da implementação do jogo *Show da Trigonometria*.

4.3 Estrutura de Diretórios

Ao se criar um projeto para aplicação Android, utilizando o IDE Android Studio, é gerado automaticamente uma estrutura de diretórios. Mais externamente encontra-se a pasta raiz, conhecida como *app*. Esta possui todos os arquivos do projeto como, por exemplo, os arquivos de codificação, as imagens utilizadas na aplicação desenvolvida e arquivos de configurações. Dentro da pasta raiz existem mais três pasta que será explicadas separadamente, utilizando como exemplo o projeto *Show Da Trigonometria*, estas pastas são elas: *manifests*, *java* e *res*. A Figura 4.2 representa a estrutura de diretórios em sua forma compactada do projeto em questão.

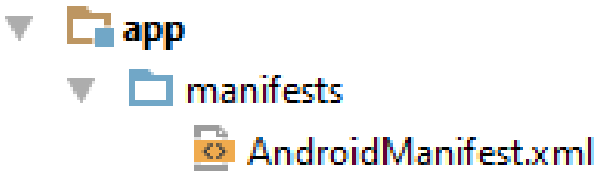
Figura 4.2 – Estrutura de diretórios em sua forma compactada



4.3.1 Pasta manifests

Dentro da pasta *manifests* está presente o arquivo de configuração *AndoridManifest.xml*, este arquivo possui as configurações essências para o funcionamento da aplicação, contendo informações como o nome do pacote da aplicação, as permissões que ela necessitará, as *activities* utilizadas entres outras importantes configurações. A Figura 4.3 ilustra a estrutura da pasta *manifests*.

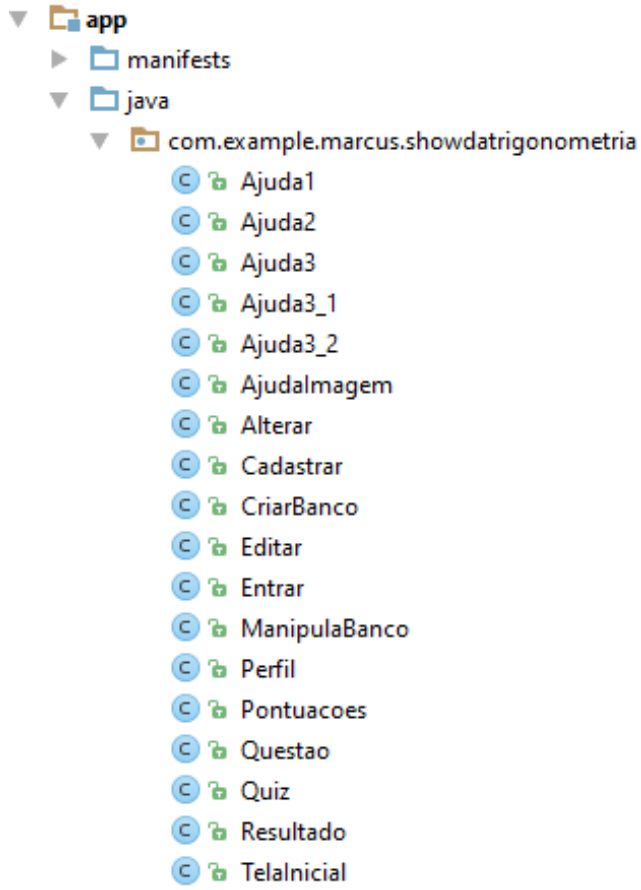
Figura 4.3 – Estrutura da pasta manifests



4.3.2 Pasta java

Na pasta *java* encontra-se o pacote padrão da aplicação com todas as classes implementadas em Java, desde as que representam as telas até as que dão suporte, como para criação e manipulação do banco de dados. A Figura 4.4 mostra a pasta *java.*

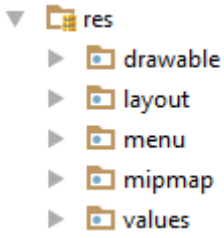
Figura 4.4 – Pasta java



4.3.3 Pasta res

Na pasta res contém os arquivos de recursos utilizadas pela aplicação, como os arquivos que estruturam o *layouts* das telas, os arquivos de internacionalização, imagens, entre outros. Esta pasta, como demonstra a Figura 4.5, está dividida em 5 pastas: *drawable*, *layout*, *menu*, *mipmap*, *values*.

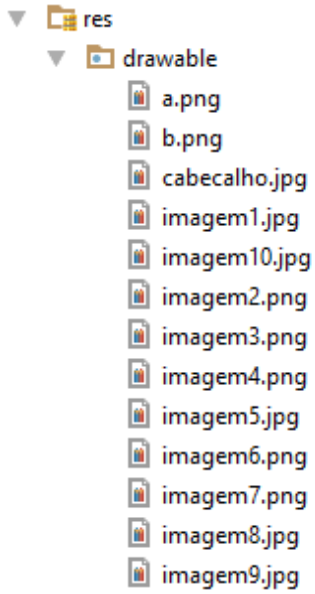
Figura 4.5 – Pasta res



- **drawable:** nesta pasta contém as imagens utilizadas na aplicação. A Figura

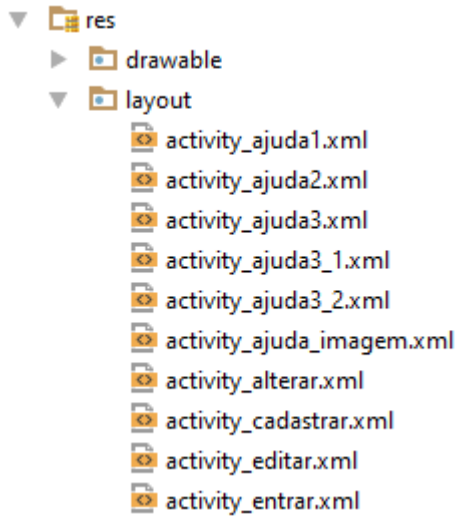
4.6 ilustra a pasta *drawable*.

Figura 4.6 – Pasta drawable



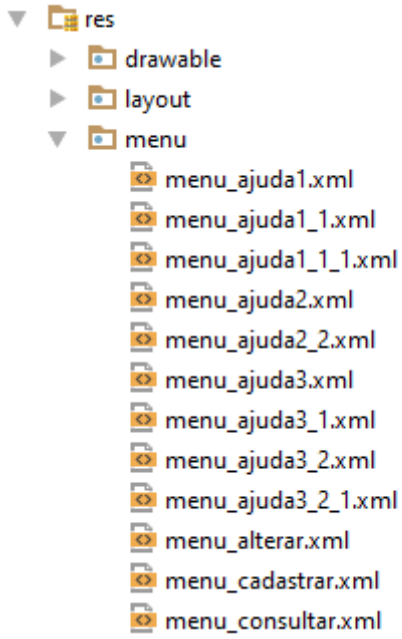
- **layout:** nesta pasta se encontra todos os arquivos de extensão XML utilizados para a estruturação do *layout* das telas da aplicação. A Figura 4.7 ilustra a pasta *layout.*

Figura 4.7 – Pasta layout



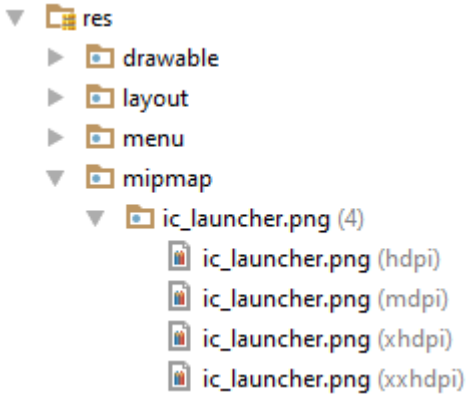
- **menu:** este diretório contém os arquivos de extensão XML utilizados para estruturação do menu de cada tela da aplicação. A Figura 4.8 ilustra a pasta *menu*.

Figura 4.8 – Pasta menu



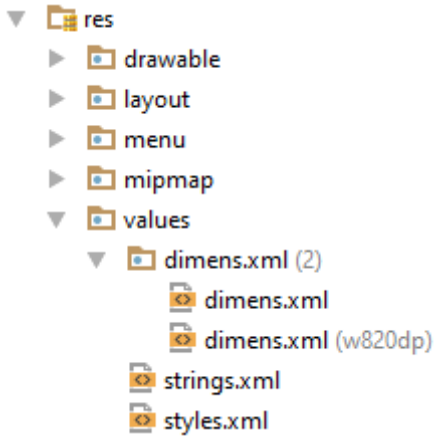
- **mipmap:** neste diretório está os ícones utilizados pela aplicação em diferentes dimensões. A Figura 4.9 demonstra a pasta *mipmap*.

Figura 4.9 – Pasta mipmap



- **values:** esta pasta contém os arquivos de extensão XML utilizados para internacionalização da aplicação e a personalização de *strings* e estilos utilizados. A Figura 4.10 mostra a pasta *values*.

Figura 4.10 – Pasta values



4.4 Classes Utilizadas

Para construção do *Show da Trigonometria* foram criadas classes para a implementação dos procedimentos que fazem a aplicação funcionar. Uma dessas classes é a *CriarBanco*, que é responsável pela criação do banco de dados com suas respectivas tabelas. Esta classe estende da classe SQLiteOpenHelper que já encapsula toda a lógica da criação. Ela e seus procedimentos serão melhor explicados na seção 4.5.2.

Outra classe criada é a *ManipulaBanco*, nesta classe estão os procedimentos para manipulação do banco de dados como, por exemplo, cadastrar, alterar e deletar um usuário. Na primeira vez que a aplicação funcionar em algum dispositivo Android, quando esta classe for chamada, ela chamará a classe *CriarBanco* e assim o banco de dados será criado.

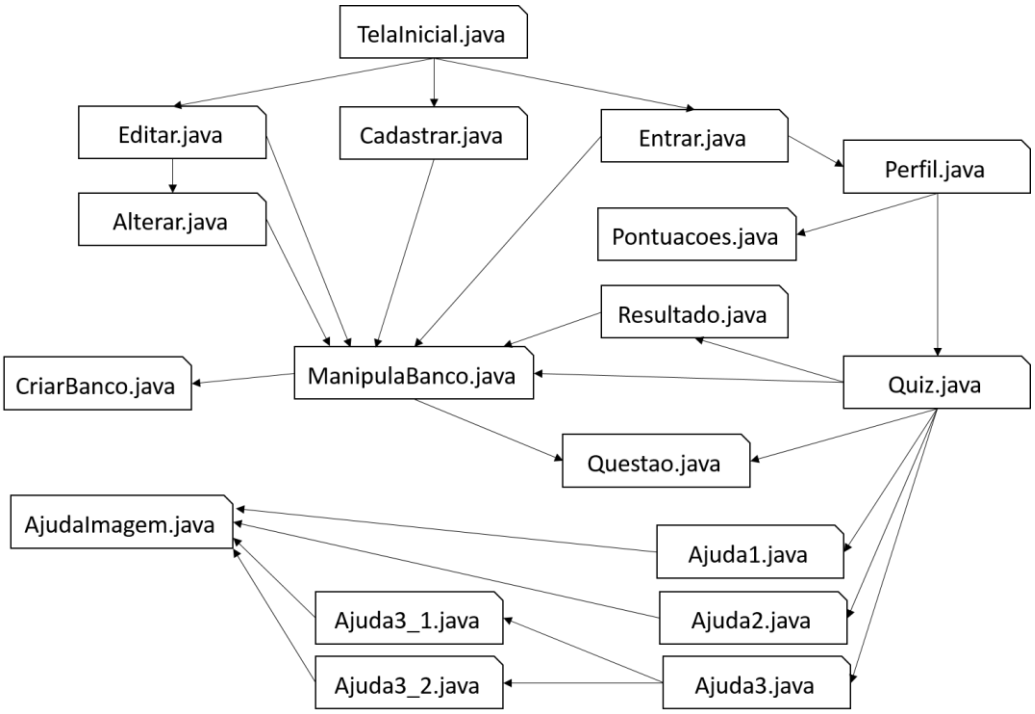
A classe *Questao* terá os procedimentos para modelar uma questão como, por exemplo, determinar para um questão as repostas possíveis a ela e a resposta correta. Essa classe é invocada pela classe *ManipulaBanco* para a inserção das perguntas no banco de dados e pela classe *Quiz* para a apresentação das perguntas e possíveis reposta ao utilizador da aplicação.

As demais classes referem-se às telas da aplicação como, por exemplo, a classe *Pontuacoes*. Ela é responsável pelos procedimentos para demonstrar ao usuário as percentagens de acertos dos modos de jogos realizados e também a classe *AjudaImagem* que contém os procedimentos para a apresentação da imagem que

servirá de ajuda para o resolução da questão.

Para uma melhor representação de como funciona as interações das classes, a Figura 4.11 traz um diagrama onde estão representadas as classes e com uma seta direcionada indica a invocação de uma outra classe como, por exemplo, na classe *TelaInicial* terá três setas direcionando para as classes *Editar*, *Cadastrar* e *Entrar*, isso indica que estando na classe *TelaInicial* é possível chamar em alguma momento/procedimento estas três classes.

Figura 4.11 – Diagrama de interações entre as classe da aplicação



4.5 Implementação

Nessa seção serão apresentadas as implementações das classes com procedimentos para o funcionamento da aplicação e da parte visual do *Show da Trigonometria* e também como se deu a criação do Banco de dados. Serão explanado separadamente cada tela da aplicação juntamente com sua respectiva classe e arquivo XML contendo a implementação dos componentes visuais das mesma. Será apresentado também as classes auxiliares como as de criação e manipulação do banco de dados.

4.5.1 Banco de dados

A criação do banco de dados para o aplicativo *Show da Trigonometria* é feita diretamente pela API Java. Foi criado a classe *CriarBanco* que herda da classe *SQLiteHelper* que já encapsula toda a lógica de criação de um banco de dados. Na classe *Criar Banco*, há a criação das duas tabelas do banco de dados. Uma das tabelas é referente ao armazenamento dos dados do usuário do sistema, como por exemplo o nome do usuário no aplicativo e seu rendimento ao realizar o jogo. A segunda tabela possui as informações sobre as perguntas, como por exemplo, a qual tema ela pertencente e a referência da imagem que irá auxiliar o entendimento da mesma. As duas tabelas criadas não tem relações entre si e suas representações lógicas são apresentadas na Seção 4.5.2.

Nesta classe se encontra os métodos onCreate (SQLiteDatabase db) e onUpgrade (SQLiteDatabase db, int versaoAntiga, int novaVersao), que são chamados automaticamente pelo Android quando o banco de dados precisa ser criado pela primeira vez ou ser atualizado devido a uma nova versão, tais métodos são apresentados na Seção 4.5.2.

4.5.2 Classe CriarBanco.java

Na classe *CriarBanco* estão os procedimentos para criação do banco de dados e suas tabelas que a aplicação utilizará. Esta classe herda da classe *SQLiteHelper*. A classe *SQLiteHelper* é uma classe auxiliar para criação do banco de dados e também para o gerenciamento da sua versão.

O banco de dados será composto por duas tabelas, uma para armazenar as informações do usuário e outra pra armazenar as informações das perguntas a serem respondias pelo usuário ao realizar o *quiz*. Inicialmente, na classe, são criados todas as variáveis que ela utiliza, como mostra a Figura 4.12. Como forma de boa prática de implementação essas variáveis foram criadas para receber o nome do banco de dados, os nomes das tabelas e os das colunas das tabelas para que, caso precise alterar algum desses campos durante o desenvolvimento ou até mesmo num trabalho futuro, não seja necessário pesquisá-los e alterá-los por todo o código da aplicação e sim, simplesmente, trocar o valor que recebem essas variáveis criadas nessa classe.

Figura 4.12 – Variáveis do banco de dados

```
protected static final String NOME_BANCO = "banco";
protected static final String TABELA = "cadastro";
protected static final String TABELA2 = "quiz";
protected static final int VERSAO = 1;
protected static final String ID = "_id";
protected static final String NOME = "nome";
protected static final String MODULO = "modulo";
protected static final String MOD1 = "mod1";
protected static final String MOD2 = "mod2";
protected static final String MOD3 = "mod3";
protected static final String TOTAL = "totalcampanha";
protected static final String NPERG = "pergondpar";
protected static final String ACERT = "acertos";
protected static final String TOTALA = "totalaleatorio";
protected static final String QUESTAO = "questao";
protected static final String RESPOSTA = "resposta";
protected static final String OPTA = "opta";
protected static final String OPTB = "optb";
protected static final String OPTC = "optc";
protected static final String OPTD = "optd";
protected static final String QMOD = "modquestao";
protected static final String IMG = "imagem";
```

A variável inteira criada de nome *versão* é utilizada para quando for preciso mudar algo na estrutura das tabelas do banco de dados depois que a aplicação já foi criada. O valor dessa variável é alterado para indicar que houve a mudança, e o método *onUpgrade*, que ainda será explanado nessa seção, será invocado automaticamente.

Para criação das tabelas que armazena as informações dos usuários e das perguntas, foram usados dois comandos *SQL*, inicialmente passados para *strings* para depois serem executados.

A *string sql* receberá o comando para criação da tabela que armazenará as informações dos usuários da aplicação. O comando possui inicialmente o texto “CREATE TABLE IF NOT EXISTS” prosseguido da variável que representa o nome da tabela, com isso a tabela com esse nome só será criada caso não existir a mesma ou outra com o mesmo nome. Logo após esse texto, entre parênteses, é determinados as colunas dessa tabela, como mostra a Figura 4.13.

Figura 4.13 – Criando a tabela cadastro

```
private static final String sql = "CREATE TABLE IF NOT EXISTS " + TABELA + " (" +
        ID + " INTEGER PRIMARY KEY AUTOINCREMENT," +
        NOME + " TEXT, " +
        MODULO + " TEXT, " +
        MOD1 + " TEXT, " +
        MOD2 + " TEXT, " +
        MOD3 + " TEXT, " +
        TOTAL + " TEXT, " +
        NPERG + " TEXT, " +
        ACERT + " TEXT, " +
        TOTALA + " TEXT); " ;
```

Vale ressaltar que as colunas, em exceção a coluna id, foram criadas do tipo *text* (texto), mesmo quando só seria trabalhado com números para essa coluna, para que pudessem receber valor *null*. A função de cada coluna para aplicação é explicado a seguir.

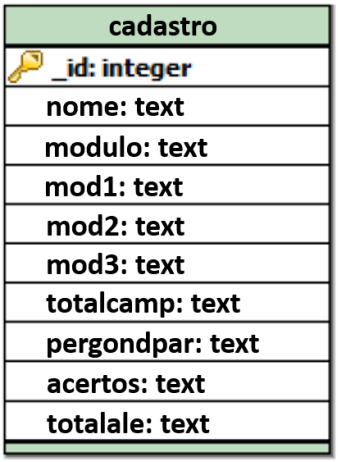
- **ID**: Essa coluna recebe o valor de *id* (identificação) do usuário cadastrado na tabela. O valor de *id* é criado automaticamente ao criar um registo e ele auto se incrementa, ou seja, a cada usuário criado o número do seu *id* será o sucessor do número do usuário registrado anteriormente a ele;
- **NOME**: Essa coluna recebe o nome escolhido pelo utilizador do sistema para seu usuário a ser criado;
- **MODULO**: Essa coluna armazena o número que referencia o módulo em que deve ser buscado as perguntas que o usuário deverá responder. As referências de cada módulo são:
    - Anglos e Arcos: 1;
    - Funções e Relações Trigonométricas: 2;
    - Triângulos: 3.
- **MOD1**: Armazena o valor da percentagem de acertos conquistados pelo usuário, na sua última realização do módulo 1;
- **MOD2**: Armazena o valor da percentagem de acertos conquistados pelo usuário, na sua última realização do módulo 2;
- **MOD3**: Armazena o valor da percentagem de acertos conquistados pelo usuário, na sua última realização do módulo 3;
- **TOTALCAMP**: Armazena a percentagem total de acertos conquistados ao realizar todos módulos possíveis do jogo, presentes no modo de jogo campanha.
- **PERGONDPAR**: Nessa coluna é salvo o valor correspondente a que pergunta o usuário irá responder ao realizar o jogo no modo aleatório. Por exemplo, se é salvo o valor 1 nessa coluna, quer dizer que o usuário está respondendo sua primeira pergunta na participação atual do *quiz* no modo aleatório, essa valor vai variar entre 1 e 15, ou seja, da primeira até a decima quinta pergunta a ser respondida. Sendo assim quando o

usuário sair sem realizar por completo a sua participação, ele terá oportunidade, de quando voltar a realizar o *quiz*, continuar a responder somente a quantidade de perguntas que faltou para terminar por completo a sua última participação;

- **ACERTOS**: Armazena a quantidades de perguntas respondidas corretamentes na realização do modo aleatório, para que possa ser feito o cálculo da percentagem de acertos total caso ocorra a participação completa do *quiz* neste modo ou para que, caso o usuário opte sair do jogo sem a realização total, possa escolher numa próxima participação continuar com o aproveitamento da sua participação anterior;
- **TOTALALE**: Armazena a percentagem total de perguntas respondidas corretamente pelo usuário no modulo aleatório. O cálculo dessa percentagem é feita sobre as 15 perguntas a serem respondidas, caso o usuário não responda todas as questões e opte em sair do jogo, ainda sim é feito o cálculo com o valor dos acertos obtido por ele na sua participação incompleta sobre as 15 perguntas necessárias para completar o *quiz*.

A Figura 4.14 exibe a representação logica da tabela cadastro.

Figura 4.14 – Representação lógica da tabela cadastro



A segunda tabela foi criada a partir da *string* sql2. Esta recebe o texto correspondente ao comando para criar a tabela referente aos registros das perguntas, como mostra a Figura 4.15. A função de cada coluna desta tabela é explicado a seguir.

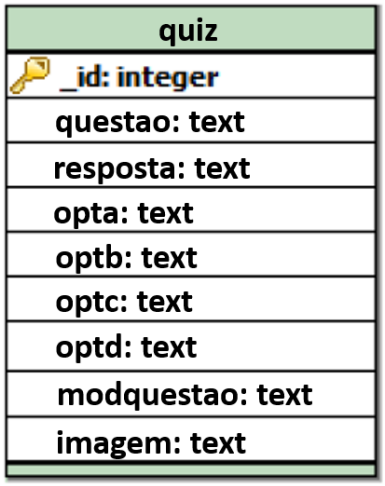
- **ID:** Essa coluna recebe o valor de *id* da pergunta cadastrada na tabela. O valor de *id* é criado automaticamente ao criar um registo e ele auto se incrementa.
- **QUESTAO:** Recebe o texto referente à pergunta a ser feita ao usuário;
- **RESPOSTA:** Armazena a resposta correta;
- **OPTA:** Armazena uma das opções de resposta oferecidas ao usuário;
- **OPTB:** Armazena uma das opções de resposta oferecidas ao usuário;
- **OPTC:** Armazena uma das opções de resposta oferecidas ao usuário;
- **OPTD:** Armazena uma das opções de resposta oferecidas ao usuário;
- **QMOD:** Recebe a referência do módulo que a pergunta pertence;
- **IMG:** Recebe o nome da imagem salva na pasta *drawable*, a qual a pergunta utilizará como complementação.

Figura 4.15 – Cirando a tabela quiz

```
private static final String sql2 = "CREATE TABLE IF NOT EXISTS " + TABELA2 + " (" +
        ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
        QUESTAO + " TEXT, " +
        RESPOSTA + " TEXT, " +
        OPTA + " TEXT, " +
        OPTB + " TEXT, " +
        OPTC + " TEXT, " +
        OPTD + " TEXT, " +
        QMOD + " INTEGER, " +
        IMG + " TEXT); " ;
```

A Figura 4.16 exibe a representação logica da tabela quiz.

Figura 4.16 – Representação lógica da tabela quiz



Dois métodos são sobrescritos nessa classe, o método *OnCreate* e o método *onUpgrade*, como mostra a Figura 4.17. O método *onCreate* é chamado de forma automática, quando a aplicação é aberta pela primeira vez. Sua tarefa é criar o banco de dados com suas devidas tabelas. O método *onUpgrade* atualiza o banco de dados de acordo com a alteração de sua versão, por exemplo, caso a aplicação seja modificada futuramente e as tabelas do banco de dados sofram alterações em suas estruturas, a versão do banco de dados será mudada para que ocorra a atualização dos dados recriando as tabelas.

Figura 4.17 – Métodos onCreate e onUpgrade

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(sql);
    db.execSQL(sql2);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABELA);
    db.execSQL("DROP TABLE IF EXISTS " + TABELA2);

    onCreate(db);
}
```

4.5.3 Classe Questao.java

Para um melhor entendimento das classes que serão abordadas posteriormente, será necessário explanar sobre a classe *Questao*. Tal classe foi criada para melhor manipular as perguntas, os conjuntos de repostas e as imagens apresentadas na realização do *quiz*.

O modelo de uma questão é composto por um texto para a pergunta, um texto para a resposta correta, quatro textos para o conjunto das possíveis respostas para essa pergunta, um número que determinará a qual módulo essa pergunta pertence e uma imagem complementar caso seja necessário.

Inicialmente na classe *Questao*, como pode ser visto na Figura 4.18, são criados o inteiro para se trabalhar com o módulo a qual as perguntas pertencem e as *strings* para o texto da pergunta, para a resposta exata dessa pergunta, para os conjuntos das respostas possíveis e para a identificação da imagem complementar.

Figura 4.18 – Criando as variáveis da classe Questao

```
public class Questao {

    private String QUESTAO;
    private String RESPOSTA;
    private String OPTA;
    private String OPTB;
    private String OPTC;
    private String OPTD;
    private int QMOD;
    private String IMG;
```

Após a declaração dos atributos, foram criados dois construtores, um sem parâmetros de entrada, sendo que os atributos da classe do tipo *String* recebem uma *string* vazia e o atributo do tipo *int* recebe valor 0. O outro construtor recebe o valor dos atributos nos parâmetros de entrada e os inicializa. Além disso, a classe contém os métodos *get* e *set* para acessar e modificar os atributos, respectivamente.

4.5.4 Classe ManipulaBanco.java

A classe *ManipulaBanco* contém as funções para se trabalhar com os registros

do banco de dados como, por exemplo, cadastrar um usuário, salvá a percentagem de acertos adquiridos ao realizar o *quiz* e adicionar as perguntas ao banco de dados caso a aplicação esteja sendo executada pela primeira vez. Inicialmente nessa classe, é criado um objeto chamado *db* da classe *SQliteDatabase*, tal classe contém os métodos de manipulação dos dados no banco, por exemplo, abrir, ler e fechar o banco de dados após sua utilização. Outro objeto criado é da classe *CriarBanco*, como mostra a Figura 4.19.

Logo após o construtor da classe, é inicializado o objeto *banco* sendo da classe *CriarBanco*.

Figura 4.19 – Objeto da classe CriarBanco

```
public class ManipulaBanco {

    private SQLiteDatabase db;
    private CriarBanco banco;

    public ManipulaBanco(Context context){
        banco = new CriarBanco(context);
    }
```

As funções da classe *ManipulaBanco* servem para facilitar as interações das outras classes da aplicação com o banco de dados. São as funções: *carregarDados, carregarDadosById, inserirRegistro, alterarRegistro, deletarRegistro, addQuestao, addQuestao(), getQuestoes, getTodasQuestoes.* Estas serão explicadas nas subseções a seguir para um melhor entendimento da classe.

4.5.4.1 Função carregarDados

A função *carregarDados* não recebe parâmetros e retornar um cursor. Nessa função é criado um objeto da classe *Cursor*, o site da Andorid (2016) explana sobre a visão geral dessa classe da seguinte forma: “Essa interface permite o acesso de leitura e escrita aleatória para o conjunto de resultados retornado por uma consulta de banco de dados”, logo esse objeto armazenará o resultado de uma consulta ao banco de dados. Também é criado um vetor de *strings* para determinar o valor de quais colunas devem ser pesquisados na consulta.

Os métodos *getReadableDatabase* e *getWiritableDatabase* serão chamados sobre o objeto *banco* para criar e/ou abrir o banco de dados para ser usado para leitura e escrita. Na primeira vez que esses métodos forem chamados, será aberto o banco de dados e seus métodos *onCreate* e *onUpgrade* serão chamados. O retorno desses métodos são um objeto válido de banco de dados, neste caso será o objeto *db*, como mostra a Figura 4.20.

Figura 4.20 – Função carregarDados

```
public Cursor carregarDados(){
    Cursor cursor;
    String[] campos =  {banco.ID, banco.NOME};

    db = banco.getReadableDatabase();
    cursor = db.query(banco.TABELA, campos, null, null, null, null, null);

    if(cursor!=null){
        cursor.moveToFirst();
    }

    return cursor;
}
```

O cursor receberá a consulta dos valores contidos das colunas, estabelecidas na *string* criada nessa função, da tabela correspondente a variável *TABELA*, ou seja, da tabela que contém os registros dos usuários.

É feita a verificação se o cursor é diferente de *null*, o que corresponde que a consulta teve algum resultado, e é usado o método *moveToFirst*, para que antes do cursor ser retornado, mover seu conteúdo para a primeira posição, fazendo assim que todos os dados possam ser mostrados.

O cursor com o resultado dessa consulta será o retorno para a função *carregarDados*.

4.5.4.2 Função carregarDadosById

A função *carregarDadosById* retorna um cursor e recebe um inteiro como parâmetro, que é tratado com o nome *id*. Esta função é usada para quando se precisa dos dados de um registro de um determinado usuário. Quando o utilizador do sistema escolher o usuário em uma lista, essa função será chamada com o *id* desse usuário escolhido, retornando assim um cursor com os dados desse usuário, como pode ser visto na Figura 4.21.

Figura 4.21 – Função carregarDadosById

```
public Cursor carregarDadoById(int id){
    Cursor cursor;
    String[] campos =  {banco.NOME, banco.MODULO, banco.MOD1,
            banco.MOD2, banco.MOD3, banco.TOTAL, banco.NPERG, banco.ACERT, banco.TOTALA};

    String where = banco.ID + "=" + id;
    db = banco.getReadableDatabase();
    cursor = db.query(banco.TABELA,campos,where, null, null, null, null,null);

    if(cursor!=null){
        cursor.moveToFirst();
    }

    return cursor;
}
```

4.5.4.3 Função inserirRegistro

A função *inserirRegistro* retorna e recebe como parâmetro uma *string*. Tal função é usada para cadastrar um usuário da aplicação do banco de dados. Nessa função é usado um objeto da classe *ContentValues*, que é utilizada para guardar um conjunto de valores como, por exemplo, ele foi usado nessa função para armazenar valores preestabelecidos para as variáveis de um novo usuário, juntamente com uma chave, tal chave será o campo correspondente a esse valor no banco de dados. A inserção dos valores e suas chaves no objeto de *ContentValues* é feito mediante o método *put***.**

Para inserir os valores no banco de dados, é utilizado o método *insertOrThow*, que é um método de adequação para inserir uma linha em uma tabela do banco de dados. O valores serão registrados no banco de dados de acordo com a palavra-chave passada que, corresponde a coluna a qual deve ser feita a inserção do valor. Caso dê algum erro para inserir um registro esse método retorna o valor -1. Logo, usando o valor de retorno do método, é estabelecida uma estrutura condicional (*if*) que caso esse método apresentar erro, será mostrado pro usuário uma mensagem de falha para inserção do registro, caso contrário uma mensagem informando sobre o sucesso da inserção é apresentada.

Como o banco de dados foi aberto para inserção de valores, é importante fechá-lo para não ocupar espaço de memória do dispositivo que está rodando a aplicação. Para tal feito é usado o método *close* sobre o objeto do banco de dados, como mostra a figura 4.22

Figura 4.22 – Função inserirRegistro

```
public String inserirRegistro(String nome){
    ContentValues valores;
    long resultado;

    db = banco.getWritableDatabase();
    valores = new ContentValues();
    valores.put(banco.NOME, nome);
    valores.put(banco.MODULO, "1");
    valores.put(banco.MOD1, "0");
    valores.put(banco.MOD2, "0");
    valores.put(banco.MOD3, "0");
    valores.put(banco.TOTAL, "0");
    valores.put(banco.NPERG, "1");
    valores.put(banco.ACERT, "0");
    valores.put(banco.TOTALA, "0");

    resultado = db.insert(banco.TABELA, null, valores);
    db.close();

    if (resultado == -1)
        return "Erro ao inserir registro";
    else
        return "Registro inserido com sucesso";
}
```

4.5.4.4 Função alterarRegistro

A função *alterarRegistro* não tem retorno e recebe como parâmetros um inteiro e *strings* que vão corresponder aos campos de um registro de usuário no banco de dados. A função será utilizada quando for necessário alterar um valor de algum registro de usuário. Essa função permite que mediante o *id* de algum usuário e o(s) campo(s) necessário(s) para alteração(ões) passados como parâmetros, sendo que para os campos do registro os quais não devem sofrer alterações devem ser passados o valor *null* como parâmetro, seja feita alteração somente dos campos desejados no registro informado.

Para fazer as devidas alterações dos campos em algum registro do banco de dados, mediante as informações da tabela, das colunas e o registro que deve ser feita as alterações é utilizado o método *update* sobre o objeto *db*, como pode ser visto na Figura 4.23.

Figura 4.23 – Função alterarRegistro

```
public void alterarRegistro(int id, String nome, String modulo, String mod1,
                            String mod2, String mod3, String total,
                            String nperg, String acert, String totala){
    ContentValues valores;
    String where;

    db = banco.getWritableDatabase();

    where = banco.ID + "=" + id;

    valores = new ContentValues();
    if (nome != null) valores.put(banco.NOME, nome);
    if (modulo != null) valores.put(banco.MODULO, modulo);
    if (mod1 != null) valores.put(banco.MOD1, mod1);
    if (mod2 != null) valores.put(banco.MOD2, mod2);
    if (mod3 != null) valores.put(banco.MOD3, mod3);
    if (total != null) valores.put(banco.TOTAL, total);
    if (nperg != null) valores.put(banco.NPERG, nperg);
    if (acert != null) valores.put(banco.ACERT, acert);
    if (totala != null) valores.put(banco.TOTALA, totala);

    db.update(banco.TABELA,valores,where,null);
}
```

4.5.4.5 Função deletarRegistro

A função *deletarRegistro*, não tem retorno e recebe como parâmetro um inteiro. Essa função terá seu uso quando for preciso apagar algum registro de usuário cadastrado no banco de dados, mediante o *id* informado, que será o inteiro passado como parâmetro. Para a remoção do registro no banco de dados é utilizado o método *delete* sobre o objeto *db* passando as informações sobre qual tabela em que será feita a remoção e o *id* do registro a ser deletado, como mostra Figura 4.28.

Figura 4.28 – Função deletarRegistro

```
public void deletarRegistro(int id){
    String where = banco.ID + "=" + id;
    db = banco.getReadableDatabase();
    db.delete(banco.TABELA, where, null);
}
```

4.5.4.6 Função addQuestao(Questao quest)

A função *addQuestao(Questao quest)* não tem retorno e recebe como

parâmetro um objeto da classe *Questao*. Essa função tem como objetivo inserir os registros das questões na tabela *quiz* do banco de dados. Cada valor das variáveis que compõe um objeto da classe *Questao* é passado a um objeto de *ContentValues* mediante a uma chave correspondente a coluna a qual o valor deve ser salvo, como mostra a Figura 4.24. Logo após, é usado o método *insert*, passando as informações de qual tabela deve ser feita a inserção e quais valores com suas respectivas colunas devem ser inseridos.

Figura 4.24 – Função addQuestao(Questao quest)

```
private void addQuestao(Questao quest) {

    db = banco.getWritableDatabase();
    ContentValues valores = new ContentValues();
    valores.put(banco.QUESTAO, quest.getQUESTAO());
    valores.put(banco.RESPOSTA, quest.getRESPOSTA());
    valores.put(banco.OPTA, quest.getOPTA());
    valores.put(banco.OPTB, quest.getOPTB());
    valores.put(banco.OPTC, quest.getOPTC());
    valores.put(banco.OPTD, quest.getOPTD());
    valores.put(banco.QMOD, quest.getQMOD());
    valores.put(banco.IMG, quest.getIMG());

    db.insert(banco.TABELA2, null, valores);
    db.close();
}
```

4.5.4.7 Função addQuestao()

A função *addQuestao*() não possui retorno e nem parâmetros. Essa função contém as informações das questões. Na função, um cursor é criado para consultar toda a tabela referente aos registros das questões. Então, é feita a verificação se não ocorreu algum erro na consulta verificando se o cursor é diferente do valor *null*. Logo após move-se o conteúdo do cursor para o seu primeiro conteúdo.

Através do método *getInt*, informando para ele um inteiro correspondente a posição do cursor, ele retornará o valor referente a essa posição, verificando se a primeira posição do cursor possui o valor 0. Isso indica se há algum conteúdo nele, para saber se a consulta presente no cursor possui algum valor, ou seja a tabela referente os registros de questões possui registros. Caso não haja nenhum conteúdo no curso é adicionado as questões no banco de dados, com isso, evitará que todas

as vezes que o utilizador do sistema for realizar o jogo, as questões tenham que ser inseridas no banco de dados.

Como dito anteriormente, caso não haja nenhum conteúdo no cursor as perguntas serão adicionadas ao banco de dados. Isto é feito da seguinte forma: primeiro é criado um objeto da classe *Questao* e é inicializado com os parâmetros referentes a pergunta, as 4 possíveis repostas, a resposta correta, o módulo a qual essa pergunta pertence e o nome salvo para a imagem complementar a essa questão caso essa seja necessária, caso a questão não possuir uma imagem complementar o valor *null* é passado no lugar do nome da imagem. Logo após essa informações serem passadas para o objeto *questao* a função *addQuestao* é chamada, que como explicado na seção anterior, tal função registrará a questão no banco de dados.

Para demonstração dessa função, a figura 4.25 mostra sua implementação com exemplos de 3 questões, cada um referenciando um módulo.

Figura 4.25 – Função addQuestao()

```
public void addQuestao() {

    db = banco.getReadableDatabase();
    Cursor cursor = db.rawQuery("SELECT count(*) FROM " + banco.TABELA2, null);
    if (cursor != null) {
        cursor.moveToFirst();
        if (cursor.getInt(0) == 0) {

            Questao q01 = new Questao(
                    " Pergunta 1 (Ufpe) Considere os triângulos retângulos PQR " +
                      "e PQS da figura a seguir. Se RS=100, quanto vale PQ?",
                          "100√3", "50√3", "50", "(50√3)/3", "50√3", 1, "a");
            this.addQuestao(q01);

            Questao q02 = new Questao(
                    " Pergunta 11 A expressão (cos²θ)/(1-senθ), com senθ≠1, é igual a ",
                      "senθ", "sen θ + 1", "tg θ . cos θ",
                                  "1", "sen θ + 1", 2, null);
            this.addQuestao(q02);

            Questao q03 = new Questao(
                    " (CESCEA-77) A soma dos catetos do triângulo retângulo." +
                           "dado que BC=10 e cos a=3/5, é:",
                              "14", "12", "10", "16", "14", 3, "img2");
            this.addQuestao(q03);

        }
    }
}
```

4.5.4.8 Função getQuestoes

A função *getQuestoes* retorna uma lista e recebe como parâmetro um inteiro. Incialmente nessa função é criado e inicializado um objeto da classe *List* possuindo sua coleção de valores do tipo *Questao*. Um objeto dessa classe é definida pelo site do Android (2010) como “uma coleção que mantém uma ordenação para seus elementos”, onde cada elemento dessa coleção possui um índice, o qual permite acessar cada elemento, lembrando que o primeiro índice será 0. Essa coleção, por ter sido criado um objeto *List*<*Questao*>, terá elementos do tipo *Questao*.

Logo após, uma *string* é criada com o nome *pegaquestao* e é inicializada com um texto referente a consulta por todos os registro da tabela *quis*, buscando os registros que possui na coluna referente ao seu módulo o mesmo valor que o inteiro passado como parâmetro. Tal aplicação é utilizada quando se precisa buscar apenas perguntas de uma determinado módulo. O resultado dessa consulta é passado para um cursor.

Essa função retornará uma lista com as questões do módulo desejado. Para isso através do conteúdo do cursor com as perguntas referente a tal módulo, são passados para objetos do tipo *Questao* os valores para suas variáveis de acordo com posição do conteúdo do cursor a que se refere. Como, por exemplo, na linha de código *quest.setQUESTAO(cursor.getString.(1))*, mostrada pela Figura 4.26, a variável *QUESTAO* do objeto *quest*, através da função *setQUESTAO*, receberá o valor contido na posição de índice 1 do cursor sendo que tal índice possui o valor referente a *string* com a pergunta que será feita no *quiz*.

Vale ressaltar que o índice de um cursor começa pelo número 0 mas tal índice através da consulta feita recebe o valor do *id* corresponde ao registro pesquisado, como não foi necessário utilizar o *id* quando aplicado essa função, não houve o registro dele no objeto criado para inserção na lista de retorno.

Figura 4.26 – Função getQUESTAO

```
public List<Questao> getQuestoes(int mod) {
    List<Questao> quesList = new ArrayList<Questao>();

    String pegaquestao = "SELECT  * FROM " + banco.TABELA2 +
            " WHERE " + CriarBanco.QMOD + " = " + mod;
    db = banco.getReadableDatabase();
    Cursor cursor = db.rawQuery(pegaquestao, null);

    if (cursor.moveToFirst()) {
        do {
            Questao quest = new Questao();
            quest.setQUESTAO(cursor.getString(1));
            quest.setRESPOSTA(cursor.getString(2));
            quest.setOPTA(cursor.getString(3));
            quest.setOPTB(cursor.getString(4));
            quest.setOPTC(cursor.getString(5));
            quest.setOPTD(cursor.getString(6));
            quest.setQMOD(cursor.getInt(7));
            quest.setIMG(cursor.getString(8));
            quesList.add(quest);
        } while (cursor.moveToNext());
    }

    return quesList;
}
```

4.5.4.9 Função getTodasQuestoes

A função *getTodasQuestoes* retorna uma lista e não recebe parâmetros. Essa função se assemelha com a função *getQuestoes*, diferenciando por de vez de buscar as questões referente a um determinado módulo, ela buscará todas as questões registradas no banco de dados. Sendo assim seu retorno será um lista contendo todas as questões armazenadas na tabela *quiz*.

Figura 4.27 – Função getTodasQuestoes

```
public List<Questao> getTodasQuestoes(){
    List<Questao> quesList = new ArrayList<Questao>();

    String pegatodas = "SELECT * FROM " + banco.TABELA2;
    db = banco.getReadableDatabase();
    Cursor cursor = db.rawQuery(pegatodas, null);

    if(cursor.moveToFirst()) {
        do {
            Questao quest = new Questao();
            quest.setQUESTAO(cursor.getString(1));
            quest.setRESPOSTA(cursor.getString(2));
            quest.setOPTA(cursor.getString(3));
            quest.setOPTB(cursor.getString(4));
            quest.setOPTC(cursor.getString(5));
            quest.setOPTD(cursor.getString(6));
            quest.setQMOD(cursor.getInt(7));
            quest.setIMG(cursor.getString(8));
            quesList.add(quest);
        } while (cursor.moveToNext());
    }

    return quesList;
}
```

4.5.5 Tela Inicial

A *Tela Inicial* do aplicativo *Show da Trigonometria*, é a primeira tela no qual o usuário vai se deparar ao abrir o jogo. Para definir uma tela como o ponto de partida da aplicação é necessário fazer certas configurações no arquivo *AndroidManifest.xml*, como pode ser visto na Figura 4.28.

Figura 4.28 – Definindo uma ponto de partida para aplicação

```
<activity
    android:name=".TelaInicial"
    android:label="Show Da Trigonometria" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

A ação *MAIN* indica que essa *activity* será a primeira a ser chamada da aplicação, vale ressaltar que apenas uma *activity* terá essa configuração. A categoria *LAUNCHER* indica que o usuário terá um ícone do aplicativo, a sua disposição, junto a outros aplicativos na tela de aplicações de seu dispositivo, como ilustrado pela Figura 4.29.

Figura 4.29 – Ícone da aplicação junto aos demais ícones

A *Tela Inicial* terá o menu principal da aplicação, definido em três botões onde que com um clique poderá ser direcionado a tela correspondente a opção escolhida, podendo ser ela o direcionamento para o *quiz*, cadastrar um usuário ou editar os usuários já cadastrados. Essa tela juntamente com seus componentes visuais é demonstrada na Figura 4.30

Figura 4.30 – Tela Inicial

4.5.5.1 Arquivo activity_inicial.XML

Para estruturação do layout da *Tela Inicial* foram usadas as configurações mostradas na Figura 4.31.

Figura 4.31 – Estruturação da Tela Inicial

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    android:gravity="center"
```

A tag principal desse XML é <*LinearLayout*>, que trata-se de um gerenciador de *layout* que organiza seus componentes visuais na vertical ou na horizontal e quando utilizado o atributo *android:orientation="vertical"* ele exibirá os componentes desse *layout* na forma de uma lista vertical. Caso fosse passado o valor *"horizontal"* os componentes desse *layout* seriam organizados horizontalmente.

Para determinar a largura e a altura de um *layout* ou de qualquer outro componente inserido na tela utiliza-se os parâmetros *android:layout_height* e *android:layout_width* que podem receber valores como *“match_parent”* que determina que estes ocuparão a tela inteira ou todo espaço disponível do *layout*, *“wrap_content*” que determina que estes ocuparão do espaço na tela ou do *layout* só o necessário para seu tamanho ou também pode ser usados números inteiros para determinar um tamanho específico e exato destes. Como pretendia-se que esse *layout* ocupasse todo o tamanho disponível da tela os valores *“match_parent”* foram passados para ambos parâmetros.

Outro parâmetro que esse *layout* utilizou foi o *android:gravity*, que determinará a disposição dos componentes do *layout*, como desejou-se que estes ficassem centralizados o valor passado foi *“center”*. Vale ressaltar que poderia, caso desejasse ou fosse necessário, passar outros valores que organizariam os componentes no *layout* a cima, ou a baixo, ou a direita, entres outros.

O parâmetro *android:paddi*ng indica que haverá um espaçamento entre a borda do *layout* e seus componentes visuais. Ouve a necessidade do uso desse parâmetro, nesse e em outros *layouts* dessa aplicação, para que os botões e outros componentes

não ficassem juntos ao canto da tela, prejudicando a estética e a forma de jogar vivenciada pelo jogador.

As configurações dos parâmetros dos botões disponíveis no *layout* da *Tela Inicial* podem ser vistas na figura 4.32.

Figura 4.32 – Criando os botões da Tela Inicial

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Entrar"
    android:id="@+id/buttonEntrar"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Cadastrar Usuário"
    android:id="@+id/buttonCadastrar"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Editar Usuário"
    android:id="@+id/buttonEditar"/>
```

Tendo, anteriormente, a explicação da funções dos parâmetros *android:layout_height* e *android:layout_width* pode notar-se que, para os botões, esses parâmetros recebem respectivamente os valores *“wrap_parent”* e *“match_parent”*, para que a largura de cada botão ocupe todo o *layout* e a altura ocupe apenas o necessário. Outro parâmetro utilizado nos botões é o *android:text* que recebe como valor o texto que o componente, que neste caso são os botões, assumirão. O parâmetro *android:id* receberá como valor um identificador para cada componente do *layout*. Vale ressaltar que cada identificador deve ser único, ou seja, cada componente receberá um identificador específico a ele. Como pode ser visto na Figura 4.32, cada botão recebeu um identificador para que, através dele, se tornasse possível manipular esses componentes nas classes que necessitará de utilizar esse arquivo XML como sua interface gráfica.

4.5.5.2 Classe TelaInicial.java

Na classe *TelaInicial* foram atribuídos as ações para os componentes da *Tela Inicial*. Primeiramente, para carregar o arquivo XML que representará a parte visual dessa tela, foi usado o método *setContentView* passando como parâmetro a referência do layout *activity_inicial* que é destinado a essa tela, como pode ser visto na Figura 4.33.

Figura 4.33 – Aplicação do método setContentView

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_inicial);
```

Após definir o *layout* a qual essa classe vai utilizar, são criados os botões na classe e através do método *findViewById* eles são vinculados aos componentes visuais criado no arquivo XML através de seus identificadores anteriormente definidos. O uso desse método pode ser visto na Figura 4.34.

Figura 4.34 – Inicializando os botões da classe TelaInicial.java

```
Button Entrar, Cadastrar, Editar;

Entrar = (Button) findViewById(R.id.buttonEntrar);
Cadastrar = (Button) findViewById(R.id.buttonCadastrar);
Editar = (Button) findViewById(R.id.buttonEditar);
```

Após isso, são definidas as ações desses botões utilizando o método *setOnClickListener*. Esse método vai controlar os eventos/ações dos botões quando eles forem acionados pelo usuário, ou seja, ao clicar no botão o método *onClick* é chamado e as ações definidas nesse métodos são inicializadas. As ações de cada botão dessa classe, consistiu-se simplesmente em trocar de telas, cada botão ao ser clicado leva a uma tela correspondente a ele. Para fazer a troca de telas foi utilizado o método *startActivity* que recebe como parâmetro uma *Intent* que é a intenção de realizar determinada tarefa, neste caso a tarefa era a troca de telas. Por exemplo, ao clicar no botão *Cadastrar* é criada a intenção de sair da *Tela Inicial* e abrir a tela *Cadastrar* chamando a classe que a representa, logo após a criação da intenção ela

é executada usando o método *startActivity*. A representação desse exemplo e as ações que serão executadas nos demais botões dessa classe podem ser vistos na Figura 4.35.

Figura 4.35 – Definindo as ações do botão da Tela Inicial

```
Entrar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(TelaInicial.this, Entrar.class);
        startActivity(intent);
    }
});

Cadastrar.setOnClickListener((v) -> {
        Intent intent = new Intent(TelaInicial.this, Cadastrar.class);
        startActivity(intent);
});

Editar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(TelaInicial.this, Editar.class);
        startActivity(intent);
    }
});
```

4.5.6 Tela Cadastrar

Na tela *Cadastrar* o usuário poderá cadastrar um novo usuário para realizar o *quiz*. Na tela se encontra um campo para inserir o nome do usuário, um botão *Cadastrar* que ao ser clicado cria um novo usuário no banco de dados com o nome inserido e um botão *Voltar* dando a opção para o usurário retornar a *Tela Inicial* sem realizar um cadastro. A representação da tela *Cadastrar* está na Figura 4.36.

Figura 4.36 – Tela Cadastrar

4.5.6.1 Arquivo activity_cadastrar.XML

O arquivo XML para representar a tela *Cadastrar* possui como gerenciador de *layout* o *LinearLayout* com a orientação vertical. Os parâmetros *android:layout_height* e *android:layout_width* recebem como valores o "*match_parent".* O parâmetro *android:padding* também é usado.

Para melhor organização dos componentes na tela, foi utilizado como um desses componentes do *layout* principal um outro *layout* contendo um *EditText*, onde esse *layout* tem como um dos seus parâmetros o *android:layout_weight* que recebe como valor um número inteiro que determina o peso desse componente no *layout* que está inserido. O parâmetro recebeu o valor 1 fazendo com que esse *layout* ocupasse toda área do *layout* principal não utilizado pelos outros componentes, que no caso são mais dois botões. Esse parâmetro pode ser usado em mais de um componente na tela, dividindo suas ocupações no *layout* inserido, de acordo com o valor do peso que o seus parâmetros recebessem. O uso do parâmetro *android:layout_weight* é ilustrado pela Figura 4.37.

Figura 4.37 – Uso do parâmetro layout_weight

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="10dp">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1" >

        <EditText

            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Nome de Usuario"
            android:id="@+id/editTextUsuario"
            android:inputType="textImeMultiLine"
            />

    </LinearLayout>
```

O *EditText* é o campo onde o usuário da aplicação pode digitar algum texto. Esse componente foi utilizado nessa tela para que o jogador pudesse digitar o nome que gostaria que tivesse seu usuário para realização do *quiz*. Os parâmetros desse *EditText* podem ser vistos na Figura 4.43. Um desses parâmetros é o *android:hint* que recebe como valor *"Nome de Usuário"*, texto que será exibido no *EditT*ext enquanto o usuário não digitar um nome no mesmo. Outro parâmetro utilizado é o *android:inputType* recebendo como valor *"textImeMultiLine"* fazendo com que a tecla *Enter* do teclado virtual do dispositivo, que ao ser clicado digitando um texto faz a quebra de linha, seja trocada pela tecla *OK*, não permitindo assim, que o jogador use a quebra de linha para digitar um nome de usuário. A comparação dos teclados virtuais do dispositivo utilizando ou não o parâmetro *android:inputType* passando o valor *"textImeMultiLine"* é demonstrada na Figura 4.38.

Figura 4.38 – Utilizando o parâmetro inputType

Os outros componentes desse *layout* são dois botões, seus respectivos parâmetros e valores podem ser vistos na Figura 4.39.

Figura 4.39 – Criando os botões da tela Cadastrar

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Cadastrar"
    android:id="@+id/bt_cadastrar"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Voltar"
    android:id="@+id/bt_voltar"/>
```

4.5.6.2 Classe Cadastrar.java

Primeiramente na classe *Cadastrar* foram criados os objetos para manipulação do banco de dados, dos componentes do *layout* com a qual a classe vai trabalhar, do texto digitado pelo usuário no campo do *EditText* e do texto para aparição de uma mensagem para o usuário. O método *onCreate* dessa classe faz a chamada do *layout activity_cadastrar* para representação visual da tela, logo após é instanciado o *CRUD* como um objeto da classe *ManipulaBanco* e ocorre a vinculação dos objetos criados para manipular os botões e o campo para digitar o nome do usuário com os componentes referentes a eles do arquivo XML, como pode ser visto na Figura 4.40.

Figura 4.40 – Inicializando os botões da classe Cadastrar

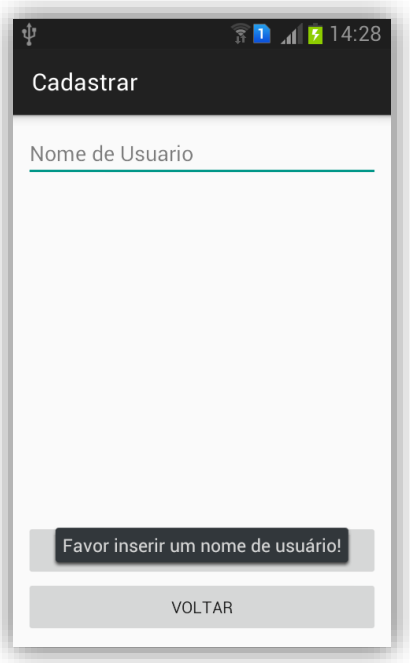
```
public class Cadastrar extends AppCompatActivity  {

    ManipulaBanco CRUD;
    Button Bt_cadastrar, Bt_voltar;
    EditText ET_usuario;
    String usuarioString, resultado;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cadastrar);

        CRUD = new ManipulaBanco(getBaseContext());
        Bt_cadastrar = (Button)findViewById(R.id.bt_cadastrar);
        Bt_voltar = (Button)findViewById(R.id.bt_voltar);
        ET_usuario = (EditText)findViewById(R.id.editTextUsuario);
```

O código da ação tomada ao clicar no botão *Cadastrar* pode ser visto na Figura 4.41. A ação implementada para o botão *Cadastrar* consiste em caso o usuário da aplicação não tenha digitado nada no campo de texto e clica no botão, uma mensagem é mostrada na tela pedindo que ele insira um nome de usuário, como pode ser visto na Figura 4.42.

Figura 4.41 – Ação do botão Cadastrar

```
Bt_cadastrar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        usuarioString = ET_usuario.getText().toString();

        if (usuarioString.equals("")) {

            Toast.makeText(getApplicationContext(),
                    "Favor inserir um nome de usuário!", Toast.LENGTH_SHORT).show();

        } else {

            resultado = CRUD.inserirRegistro(usuarioString);
            Toast.makeText(getApplicationContext(), resultado, Toast.LENGTH_SHORT).show();
            finish();
        }
    }
});
```

Figura 4.42 – Mensagem solicitando um nome de usuário



Caso o usuário da aplicação digite um nome no campo de texto e clica no botão *Cadastrar* será chamada a função *inserirRegistro* da classe *ManipulaBanco* mostrando um mensagem na tela com o retorno da função e a tela *Cadastrar* irá ser finalizada pelo método *finish*, como pode ser visto na Figura 4.43.

Figura 4.43 – Mensagem confirmando o cadastro

Na classe *Cadastar* ainda foi implementado a ação para o botão *Voltar*, que simplesmente finaliza a tela *Cadastrar* usando o método *finish*. O código da ação para esse botão pode ser visto na Figura 4.44.

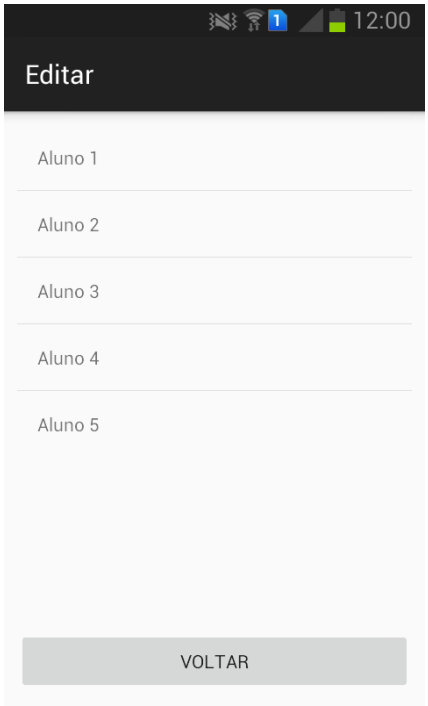
Figura 4.44 – Ação do botão Voltar da tela Cadastrar

```
Bt_voltar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        finish();
    }
});
```

4.5.7 Tela Editar

Na tela *Editar* o utilizador do aplicativo terá um lista com os nomes de todos os usurários cadastrados na aplicação, clicando em algum nome de usuário da lista abrirá a tela *Alterar*. Na tela *Editar* também terá um botão *Voltar*, que permitirá retornar a *Tela Inicial* sem necessário fazer alguma edição. A disposição visual da tela *Editar* pode ser vista Figura 4.45.

Figura 4.45 – Tela Editar



4.5.7.1 Arquivo activity_editar.XML

O arquivo XML que representa a tela *Editar* possui como gerenciador de *layout* o *LinearLayout* com a orientação vertical. *"match_parent"* é passado como valor aos parâmetros *android:layout_height* e *android:layout_width*. Também foi utilizado o parâmetro *android:padding*.

Na tela *Editar*, como na tela *Cadastrar*, foi utilizado como um componente do *layout* principal um outro *layout*, para um melhor organização e disposição visual da tela. Esse componente *layout* possui o *ListView*, que é um componente de interface gráfica utilizado para listar dados, no caso os usuários cadastrados na aplicação. Como parâmetros o *ListView* recebeu o *layout_width* e o *layout_height* com os valores *match_parent* e *wrap_content* respectivamente, e o parâmetro *id* para definir uma identificação para o *ListView* ser manipulado na classe que utilizar esse arquivo XML.

Outro componente desse *layout* é um botão, seus parâmetros e dos outros componentes do arquivo *activity_editar.XML* podem ser vistos na Figura 4.46.

Figura 4.46 – Componentes do arquivo activity_editar.XML

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="10dp" >

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" >

        <ListView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/listView" />


    </LinearLayout>

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Voltar"
        android:id="@+id/bt_voltar"
        android:layout_gravity="center_horizontal" />


</LinearLayout>
```

4.5.7.2 Classe Editar.java

Na classe *Editar*, como pode ser visto na Figura 4.47, foram criados um objeto para manipulação do banco de dados, um objeto para manipular uma lista, um objeto para manipular um botão e um inteiro chamado código. Para representação visual da tela é chamado o arquivo *activity_editar* no método *setContentView*. Logo após o objeto para manipular o botão e o objeto para manipular a lista encontrados nessa tela são vinculados aos componentes referentes a eles do arquivo XML, também é instanciado o *CRUD* como um objeto da classe *ManipulaBanco*.

Figura 4.47 – Inicializando os objetos da classe Editar

```
public class Editar extends AppCompatActivity {

    ManipulaBanco CRUD;
    ListView Lista;
    Button Bt_voltar;
    int codigo;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_editar);

        Bt_voltar = (Button) findViewById(R.id.bt_voltar);
        Lista = (ListView) findViewById(R.id.listView);
        CRUD = new ManipulaBanco(getBaseContext());
```

Nessa classe também é criado um objeto da classe *Cursor*, como pode ser visto na Figura 4.48. O objeto criado de nome *Cursor* receberá o retorno da função *carregarDados*, que como explicado anteriormente, tal retorno é um cursor que recebe uma consulta de todos os *id*'s e seus respectivos nomes de usuário cadastrados na aplicação.

Figura 4.48 – Inicializando o objeto Cursor

```
final Cursor Cursor = CRUD.carregarDados();
```

Para o preenchimento dos dados da lista da tela *Editar*, foi criado um objeto, cujo nome é adaptador, da classe *SimpleCursorAdapter*, que servem como

adaptadores que recebem dados de um cursor e coloca-os em seus respectivos lugares no *layout* a ser usado.

Cada item da lista da tela *Editar* possui um *layout* que é um dos parâmetros recebidos pelo construtor da *SimpleCursorAdapter*, esse *layout* possui apenas um componente que é um *TextView* como pode ser visto na Figura 4.49.

Figura 4.49 – *Layout* da lista da tela Editar

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingTop="10dp"
    android:paddingBottom="15dp"
    android:paddingRight="10dp"
    android:paddingLeft="10dp">
    <TextView
        android:id="@+id/nomeUsuario"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="5dp"
        android:layout_marginTop="5dp"
        android:text="NOME" />

</LinearLayout>
```

Os parâmetros necessários a serem passados para o *SimpleCursorAdapter*, como pode ser visto na Figura 4.50, primeiramente é o *context,* o segundo é o *layout* de cada item, o terceiro parâmetro é o cursor com os dados a serem mostrados, o quarto é um *array* com as colunas do cursor que serão mostradas, o quinto parâmetro é um *array* de mesmo tamanho que o do anterior com os elementos que receberão os dados.

Logo após é passado para a lista da tela *Editar* o objeto *adaptador* através do método *setAdapter*, fazendo com que essa lista seja preenchida com os itens contidos neste objeto.

Figura 4.50 – Utilizando o método setAdapter

```
String[] nomeCampos = new String[]{CriarBanco.NOME};
int[] idViews = new int[]{R.id.nomeUsuario};

SimpleCursorAdapter adaptador = new SimpleCursorAdapter(getBaseContext(),
        R.layout.usuario_nome, Cursor, nomeCampos, idViews);

Lista.setAdapter(adaptador);
```

Para definir a ação de se clicar em um item da lista é o utilizado o método *setOnItemClickListener*, esse método vai controlar os eventos/ações dos itens da lista quando eles forem clicados pelo usuário, ou seja, ao clicar em um item da lista o método *onItemClick* é chamado e as ações definidas nesse métodos são inicializadas.

Ao clicar em um item da lista, que no caso será o nome de um usuário, o inteiro criado no início da classe, que possui o nome de *codigo*, receberá o *id* correspondente a este nome de usuário no banco de dados. É criado uma *Intent* que efetuará a troca de telas entre a *Editar* para a *Alterar*, e através do método *putExtra* o inteiro *codigo* será passado como parâmetro para a classe referente a tela *Alterar*, como pode ser visto na Figura 4.51.

Para a utilização do método putExtra é necessário dois parâmetros o primeiro será uma *string* que é espécie de chave de acesso para o segundo parâmetro que é o valor a ser passado para uma outra tela, caso quando utilizar o valor passado este será referenciado pela sua chave de acesso.

Figura 4.51 – Passagem de parâmetros com o método putExtra

```
Lista.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

        Cursor.moveToPosition(position);
        codigo = Cursor.getInt(Cursor.getColumnIndexOrThrow(CriarBanco.ID));
        Intent intent = new Intent(Editar.this, Alterar.class);
        intent.putExtra("codigo", codigo);
        startActivity(intent);
        finish();
    }
});
```

Também na classe *Editar* foi implementado a ação para o botão *Voltar*, que finalizará a tela *Editar* usando o método *finish*, como pode ser visto na Figura 4.52.

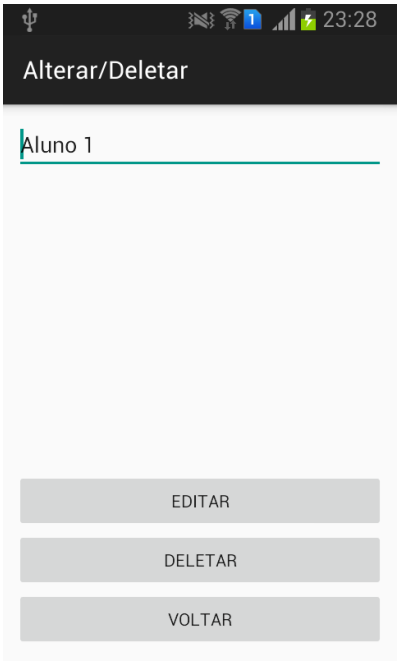
Figura 4.52 – Finalizando a tela Editar

```
Bt_voltar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        finish();
    }
});
```

4.5.8 Tela Alterar/Deletar

Na tela *Alterar/Deletar* o utilizador da aplicação poderá efetuar a edição do nome de um usuário já cadastrado. Nessa tela ele encontrará um campo de edição de texto com o nome de usuário a qual se quer alterar, podendo fazer nesse campo as alterações que se julgar necessária a este nome, um botão *Editar* para confirmação da edição do nome do usuário, um botão *Deletar* para apagar o registro desse usuário do banco de dados e um botão para retornar a página anterior. A tela *Alterar/Deletar* é representada pela Figura 4.53.

Figura 4.53 – Tela Alterar/Deletar



4.5.8.1 Arquivo activity_alterar.XML

O arquivo XML *activity_alterar* possui como gerenciador de *layout* o *LinearLayout* com a orientação vertical. Para os parâmetros *android:layout_height* e *android:layout_width* o *"match_parent"* é passado como valor.

Um dos componentes desse *layout* principal é um outro *layout* que possui como componente um *EditText* que possui como uns dos seus parâmetros o *android:hint* recebendo como valor *"Nome de Usuário"*, este texto aparecerá caso o utilizador da aplicação apague todo o nome o qual se quer alterar. Outros explorados parâmetros

são o *inputType* e o *id*. Tais componentes e seus parâmetros podem ser melhores visualizados pela Figura 4.54.

Figura 4.54 – Componentes do arquivo activity_alterar.XML

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:padding="10dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1">

        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Nome de Usuario"
            android:id="@+id/editTextUsuarioED"
            android:inputType="textImeMultiLine"/>
    </LinearLayout>
```

Outros componentes desse *layout* são três botões, seus parâmetros podem ser vistos na Figura 4.55.

Figura 4.55 – Outros componentes do arquivo activity_alterar.XML

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Editar"
    android:id="@+id/bt_editar"
    android:layout_below="@+id/editTextUsuarioED"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Deletar"
    android:id="@+id/bt_deletar"
    android:layout_below="@+id/bt_editar"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Voltar"
    android:id="@+id/bt_voltar"
    android:layout_below="@+id/bt_deletar"/>
```

4.5.8.2 Classe Alterar.java

Para a classe *Alterar* foram criados objeto para manipulação do banco de dados, objetos para manipulação dos componentes do *layout* com a qual a classe vai utilizar, um inteiro que receberá o parâmetro passado pela troca de telas e um cursor. Para a representação visual o arquivo *activity_alterar* é chamado, logo após, o inteiro criado recebe através do método *getIntExtra* o parâmetro vindo da tela anterior, a tela *Editar*, tal parâmetro é o *id* do usuário escolhido para ser editado. O *CRUD* é instanciado como um objeto da classe *ManipulaBanco* e ocorre a vinculação dos objetos criados para manipular os botões *Voltar*, *Cadastrar* e *Deletar* e o campo para editar o nome do usuário com os componentes referentes a eles do arquivo XML, como pode ser visto na Figura 4.56.

Figura 4.56 – Criando e inicializando objetos da classe Alterar

```
public class Alterar extends AppCompatActivity {

    EditText Nome;
    Button Bt_editar, Bt_deletar, Bt_voltar;
    ManipulaBanco CRUD;
    Cursor Cursor;
    int codigo;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_alterar);

        codigo = this.getIntent().getIntExtra("codigo", 0);
        CRUD = new ManipulaBanco(getBaseContext());
        Nome = (EditText)findViewById(R.id.editTextUsuarioED);
        Bt_editar = (Button)findViewById(R.id.bt_editar);
        Bt_deletar = (Button)findViewById(R.id.bt_deletar);
        Bt_voltar = (Button)findViewById(R.id.bt_voltar);
```

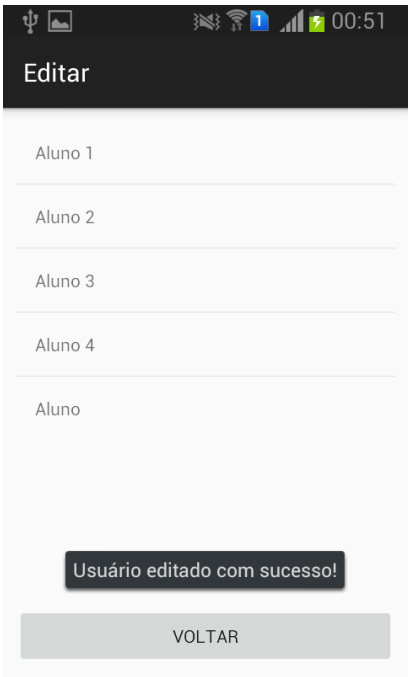
O cursor receberá o retorno da função *carregaDadoById* passando como parâmetro o inteiro correspondente o *id* do usuário escolhido para edição, o *EditText* do *layout* será inicializado com nome deste usuário, para tal feito usasse o método *getString* e *getColumnIndexOrThrow* passando como parâmetro a coluna *NOME* para que se possa pegar o texto correspondente a esta no cursor que foram carregados os dados do usuário. A utilização de tais métodos podem ser vistos na Figura 4.57.

Figura 4.57 – Utilizando os métodos getString e getColumnIndexOrThrow

```
Cursor = CRUD.carregarDadoById(codigo);
Nome.setText(Cursor.getString(Cursor.getColumnIndexOrThrow(CriarBanco.NOME)));
```

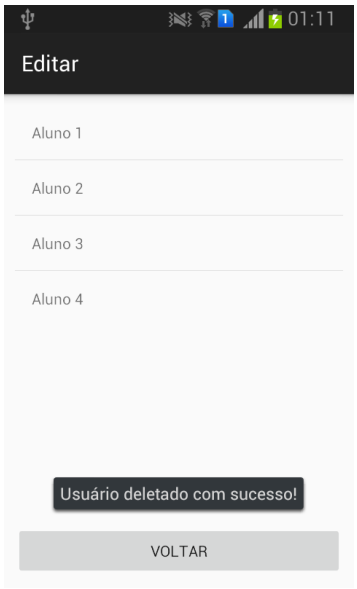
Para o botão *Editar*, foi implementado a ação de chamar a função *alterarRegistro* da classe *ManipulaBanco* passando como parâmetro o inteiro com o *id* do usuário escolhido pra edição e o texto digitado pelo utilizador da aplicação no campo *EditText*. Acionando o botão haverá a aparição da mensagem “Usuário editado com sucesso!”, como pode ser visto na Figura 4.58. Caso o utilizador da aplicação remova totalmente o texto do *EditText* a mensagem "Favor inserir um nome de usuário!" é mostrada. Ao clicar no botão *Editar* também ocorre, através do método *Intent,* o retorno a tela *Editar*.

Figura 4.58 – Mensagem apresentada ao editar o usuário



Para o botão *Deletar*, foi implementado o evento de chamar a função *deletarRegistro* da classe *ManipulaBanco* passando como parâmetro o inteiro com o *id* do usuário escolhido pra edição fazendo assim a remoção deste no banco de dados e ocorre a demonstração da mensagem “Usuário deletado com sucesso!”, como pode ser visto na Figura 59. Através do método *Intent* se retorna a tela *Editar*.

Figura 4.59 – Mensagem apresentada ao deletar o usuário



Para a ação do botão *Voltar*, foi implementado o método *Intent* para retornar à tela *Editar* e o método *finish* para encerrar a tela *Alterar/Deletar*. O código da ação para esse e os demais botões dessa classe pode ser visto na Figura 4.60.

Figura 4.60 – Ações dos botões da classe Alterar

```
Bt_editar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (Nome.getText().toString().equals("")) {
            Toast.makeText(getApplicationContext(),
                    "Favor inserir um nome de usuário!", Toast.LENGTH_SHORT).show();
        } else {

            CRUD.alterarRegistro(codigo, Nome.getText().toString(),
                    null, null, null, null, null, null, null, null);
            Toast.makeText(getApplicationContext(),
                    "Usuário editado com sucesso!", Toast.LENGTH_LONG).show();
            Intent intent = new Intent(Alterar.this, Editar.class);
            startActivity(intent);
            finish();
        }
    }
});
Bt_deletar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        CRUD.deletarRegistro(codigo);
        Toast.makeText(getApplicationContext(),
                "Usuário deletado com sucesso!", Toast.LENGTH_LONG).show();
        Intent intent = new Intent(Alterar.this, Editar.class);
        startActivity(intent);
        finish();
    }
});
Bt_voltar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(Alterar.this, Editar.class);
        startActivity(intent);
        finish();
    }
});
```

Na classe *Alterar* também foi feita a sobrescrita da ação da tecla “retornar” dos dispositivos Android, tal tecla é ilustrada na Figura 4.61 e destacada pelo círculo.

Figura 4.61 – Tecla “retornar”

Para a sobrescrita desta ação é usado o *@Override* no método *onBackPressed*, que permitirá sobrescrever o evento, que já é determinado pela plataforma Android para esse método. Agora ao acionar essa tecla, abrirá a tela *Editar* e finalizará a tela *Alterar*, como pode ser visto na Figura 4.62.

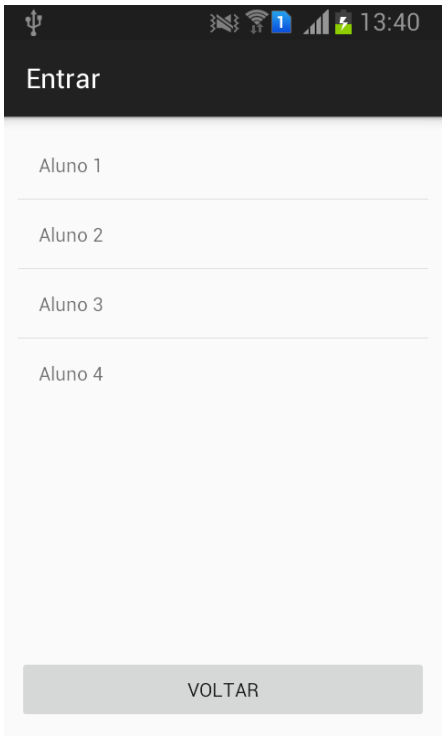
Figura 4.62 – Sobrescrevendo a ação da tecla “retonar”

```
@Override
public void onBackPressed() {
    Intent intent = new Intent(Alterar.this, Editar.class);
    startActivity(intent);
    finish();
}
```

4.5.9 Tela Entrar

Na tela *Entrar* o utilizador do aplicativo terá um lista com os nomes de todos os usurários cadastrados na aplicação, deverá ser escolhido um usuário da lista para realização do *quiz* clicando sobre ele. Na tela *Entrar* também terá um botão *Voltar*, que permitirá retornar a *Tela Inicial*. A disposição visual da tela *Entrar* pode ser visualizada na Figura 4.63.

Figura 4.63 – Tela Entrar



4.5.9.1 Arquivo activity_entrar.XML

O arquivo *activity_entrar* possui como gerenciador de *layout* o *LinearLayout* com a orientação vertical. *“match_parent”* é passado como valor aos parâmetros *android:layout_height* e *android:layout_width*. Também foi utilizado o parâmetro *android:padding*.

Como componentes do *layout* principal tem um *layout* com uma *ListView* e um botão, os componentes e seus parâmetros podem ser melhor visualizados na Figura 4.64.

Figura 4.64 – Componentes do arquivo activity_entrar

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp" >

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" >

        <ListView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/listView"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true" />
    </LinearLayout>

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Voltar"
        android:id="@+id/bt_voltar"
        android:layout_gravity="center_horizontal" />

</LinearLayout>
```

4.5.9.2 Classe Entrar.java

Na classe *Entrar*, como pode ser visto na Figura 4.65, foram criados um objeto para manipulação do banco de dados, um objeto para manipular uma lista, um objeto para manipular um botão, um cursor e um inteiro chamado código. Para representação visual da tela é chamado o arquivo *activity_entrar* no método *setContentView*. Logo após, o objeto para manipular o botão e o objeto para manipular a lista encontrados na tela *Entrar* são vinculados aos componentes referentes a eles do arquivo XML, também é instanciado o *CRUD* como um objeto da classe *ManipulaBanco*.

Figura 4.65 – Criando e inicializando os objetos do aquivo activity_entrar

```
ManipulaBanco CRUD;
ListView Lista;
Button Bt_voltar;
int codigo;
Cursor Cursor;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_entrar);

    Bt_voltar = (Button) findViewById(R.id.bt_voltar);
    Lista = (ListView) findViewById(R.id.listView);
    CRUD = new ManipulaBanco(getBaseContext());
```

O objeto criado de nome *Cursor* receberá o retorno da função *carregarDados*. Para o preenchimento dos dados da lista da tela *Entrar*, foi criado um objeto da classe *SimpleCursorAdapter* com o nome *adaptador*. Após ser fornecido os devidos parâmetros ao *SimpleCursorAdapter* foi passado para a lista da tela *Entrar* o objeto *adaptador*, sendo assim a lista foi preenchida com os itens contidos neste objeto.

Ao se clicar em um item da lista o inteiro criado no início da classe, que possui o nome de *codigo*, receberá o *id* correspondente ao nome de usuário escolhido. É criado uma *Intent* que efetuará a troca de telas entre a *Entrar* para a *Perfil*, passando o inteiro *codigo* como parâmetro, como pode ser visto na Figura 4.66.

Figura 4.66 – Passando o id como parâmetros entres telas

```
final Cursor Cursor = CRUD.carregarDados();

String[] nomeCampos = new String[]{CriarBanco.NOME};
int[] idViews = new int[]{R.id.nomeUsuario};

SimpleCursorAdapter adaptador = new SimpleCursorAdapter(getBaseContext(),
        R.layout.usuario_nome, Cursor, nomeCampos, idViews, 0);

Lista.setAdapter(adaptador);

Lista.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

        Cursor.moveToPosition(position);
        codigo = Cursor.getInt(Cursor.getColumnIndexOrThrow(CriarBanco.ID));

        Intent intent = new Intent(Entrar.this, Perfil.class);
        intent.putExtra("codigo", codigo);
        startActivity(intent);
    }
});
```

4.5.10 Tela Perfil

Na tela *Perfil* o utilizador da aplicação irá se deparar com o nome do usuário que se pretende realizar o *quiz* selecionado na página *Entrar*. Além deste pode ser encontrado quatros botões. Um botão com o nome *MODO CAMPANHA* para resolver o *quiz* de forma a responder grupos de perguntas sequencialmente. Outro botão com o nome *MODO ALEATÓRIO* para jogar o *quiz* de forma a responder perguntas aleatoriamente variando ou não o tema da pergunta posterior a respondida. Os outros dois são, um botão de nome *PONTUAÇÕES* para ir para a tela onde poderá ver a performance o responder as quetões do *quiz* até o momento nos dois tipos de modo de jogar e um botão voltar que possibilita retornar a página *Entrar*.

Figura 4.67 – Tela Perfil

4.5.10.1 Arquivo activity_perfil.XML

O arquivo *activity_perfil* possui como gerenciador de *layout* o *LinearLayout* com a orientação vertical. Os parâmetros *android:layout_height*, *android:layout_width* e *android:padding* também são usados nesse *layout*.

Um dos componentes do *layout* principal é um *TextView* tendo como um dos

seus parâmetros o *textSize* que permite alterar o tamanho da fonte do texto e o *textStyle* que permite alterar o estilo da fonte do texto como, por exemplo, se passado *"bold"* como valor para o parâmetro *textStyle*, o texto desse *textView* ficará em negrito. A declaração desses parâmetros pode ser vista na Figura 4.68.

Figura 4.68 – Utilizando os parâmetros testSize e textStyle

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/nome"
        android:layout_gravity="center"
        android:textSize="20dp"
        android:textStyle="bold"
        />
```

Outros componentes desse *layout* principal, como pode ser visto na Figura 4.69, é um *layout* com três botões e mais um botão.

Figura 4.69 – Outros componentes do arquivo activity_perfil

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_weight="1"
    android:gravity="center">

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Modo Campanha"
    android:id="@+id/buttonCampanha" />

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Modo Aleatório"
    android:id="@+id/buttonAleatorio" />

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Pontuações"
    android:id="@+id/buttonPontuacoes" />

</LinearLayout>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/voltar"
    android:text="Voltar"/>
```

4.5.10.2 Classe Perfil.java

Para a classe *Perfil* foram criados objetos para manipular os botões e um *TextView*. Também foi criado um objeto para manipulação do banco de dados, três inteiros cujo os nomes são *nmodulo*, *codigo* e *nperg*. Um objeto para manipulação de um cursor e um *AlertDialog*, que será melhor explicado posteriormente, também foram criados. Para representação visual da tela é chamado o arquivo *activity_perfil* no método *setContentView*. Logo após os objetos para manipulação dos botões e do *TextView* são vinculados aos componentes referentes a eles do arquivo XML. Também é instanciado o *CRUD* como objeto da classe *ManipulaBanco*.

Figura 4.70 – Criando e inicializando os objetos da classe Perfil

```
Button bt_modoAleatorio, bt_modoCampanha, bt_pontuacoes, bt_voltar;
AlertDialog VerificaJogo;
int nmodulo, codigo, nperg;
ManipulaBanco CRUD;
Cursor cursor;
TextView tv;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_perfil);

    bt_modoAleatorio = (Button) findViewById(R.id.buttonAleatorio);
    bt_modoCampanha = (Button) findViewById(R.id.buttonCampanha);
    bt_pontuacoes = (Button) findViewById(R.id.buttonPontuacoes);
    bt_voltar = (Button) findViewById(R.id.voltar);
    tv = (TextView) findViewById(R.id.nome);

    CRUD = new ManipulaBanco(getBaseContext());
```

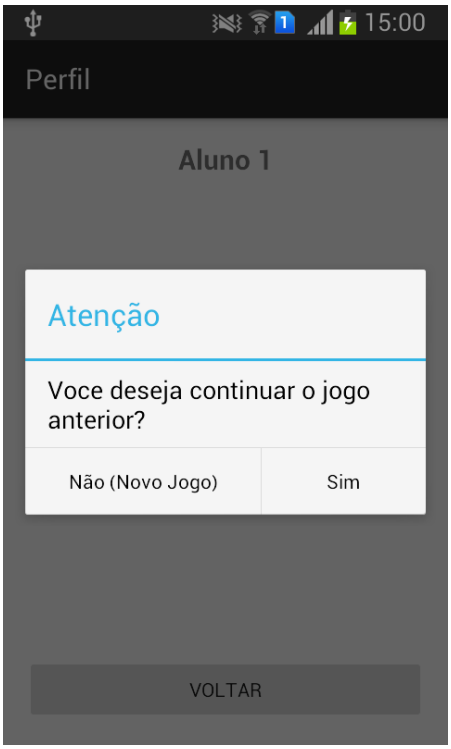
O inteiro *codigo* recebe o valor passado como parâmetro entre telas, contendo o *id* do usuário escolhido na tela *Entrar*. O cursor recebe a consulta do banco de dados provinda da função *carregarDadoById* passando como parâmetro o inteiro *codigo*. Logo após ter no cursor os dados do usuário escolhido para dar sequência do *quiz* pelo utilizador da aplicação, o *TextView* recebe o nome do usuário, o inteiro *nmodulo* e o inteiro *nperg* receberão o valor do módulo e do número de perguntas desse usuário respectivamente, como pode ser visto na Figura 4.71.

Figura 4.71 – Inicializando os inteiro nmodulo e nperg

```
codigo = this.getIntent().getIntExtra("codigo", 0);
cursor = CRUD.carregarDadoById(codigo);

tv.setText(cursor.getString(cursor.getColumnIndexOrThrow(CriarBanco.NOME)));
nmodulo = cursor.getInt(cursor.getColumnIndexOrThrow(CriarBanco.MODULO));
nperg = cursor.getInt(cursor.getColumnIndexOrThrow(CriarBanco.NPERG));
```

No método *OnClick* do botão correspondente ao modo campanha tem uma estrutura condicional (*if*) que caso o usuário em questão tenha parado um jogo sem finalizar todo o *quiz*, encerrando assim sua participação em um módulo posterior ao primeiro, aparecerá um caixa de diálogo, conhecida na programação para dispositivos Android como *AlertDialog*. Essa caixa de diálogo apresentará a pergunta “Você deseja continuar o jogo anterior?”, como pode ser visto na Figura 4.72.

Figura 4.72 – Utilizando caixa de diálogo



O utilizador da aplicação terá que tomar a decisão entre duas opções, a primeira é “Sim”, sendo que caso for escolhida, será chamada a classe referente tela *Quiz* passando como parâmetro o *id* do usuário e o valor 1, referente ao modo campanha. Caso seja escolhida a opção “Não (Novo Jogo)” a função *alterarRegistro* é chamada alterando o módulo desse usuário novamente para o primeiro e zerando

suas pontuações, a classe referente a tela *Quiz* é chamada recebendo os valores do *id* e o modo de jogo escolhido como parâmetros entre trocas de telas.

A declaração desse *AlertDialog* está representada pela figura 4.73. Como pode ser notado visualizando essa figura, através do método *Builder* pode-se alterar as mensagens e as ações dos botões presentes no *AlertDialog* e, através dos métodos *create* e *show*, a caixa de diálogo possa ser criada e mostrada ao utilizador da aplicação.

Figura 4.73 – Declarando o AlertDialog

```
bt_modoCampanha.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (nmodulo != 1) {
            AlertDialog.Builder adBuilder = new AlertDialog.Builder(Perfil.this);
            adBuilder.setTitle("Atenção");
            adBuilder.setMessage("Voce deseja continuar o jogo anterior?");
            adBuilder.setCancelable(true);
            adBuilder.setPositiveButton("Sim", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    Intent intent = new Intent(Perfil.this, Quiz.class);
                    intent.putExtra("codigo", codigo);
                    intent.putExtra("modo", 1);
                    startActivity(intent);
                    finish();
                }
            });
            adBuilder.setNegativeButton("Não (Novo Jogo)", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    CRUD.alterarRegistro(codigo, null, "1", "0", "0", "0", "0", null, null, null);
                    Intent intent = new Intent(Perfil.this, Quiz.class);
                    intent.putExtra("codigo", codigo);
                    intent.putExtra("modo", 1);
                    startActivity(intent);
                    finish();
                }
            });
            VerificaJogo = adBuilder.create();
            VerificaJogo.show();
        } else {
            Intent intent = new Intent(Perfil.this, Quiz.class);
            intent.putExtra("codigo", codigo);
            intent.putExtra("modo", 1);
            startActivity(intent);
            finish();
        }
    }
});
```

Outro método que pode ser utilizado numa caixa de diálogo é o *setCancelable* que, quando recebido o valor *“TRUE”*, permite fechar a caixa de diálogo usando a tecla “*retornar”*, caso receba valor *“FALSE”* quando a tecla *“retornar”* for acionada não fechará a caixa de diálogo.

Vale observar que, a tela onde serão apresentadas as perguntas é a mesma para os dois modos de jogo oferecidos pela aplicação, o que vai diferenciar os modos

é o valor passado como parâmetro para tela do *quiz*, sendo que ficou definido o modo campanha como valor 1 e o modo aleatório como valor 2.

Como nota-se na Figura 4.73, caso o usuário em questão não tenha jogado o *quiz* ainda ou tenha completado totalmente na sua última participação, a classe referente tela Quiz é chamada, passando como parâmetro na troca de telas o valor do *id* desse usurário e do módulo escolhido.

No método *onClick* do botão referente ao modo aleatório, como pode ser visto na Figura 4.74, tem uma estrutura condicional (*if*), caso o usuário escolhido para jogar o *quiz* tem em seu registro um último jogo não finalizado, ou seja, ter respondidos um número menor que 15 perguntas na sua última participação no *quiz*, aparecerá uma caixa de diálogo onde terá que decidir em responder o número de perguntas para completar o *quiz* que faltou em sua última participação ou recomeçar o jogo. Caso escolha a opção para continuar o jogo anterior, a classe referente tela *Quiz* é chamada, sendo passado como parâmetros da troca de telas o *id* do usuário em questão e o valor do modo de jogo escolhido. Mas, caso for escolhido a opção de começar um novo jogo, a função *alterareRegistro* é chamada fazendo com que modifique o valor do número de perguntas respondidas, zere a quantidade de acertos e a percentagem referente ao modo aleatório.

Se o usuário esteja jogando o *quiz* pela primeira vez ou tenha finalizado completamente as 15 perguntas na sua última participação, a classe referente à tela *Quiz* é chamada e os valores do *id* do usuário em questão e do modo de jogo escolhido são transferidos para ela, não sendo feito o uso de uma caixa de diálogo.

Figura 4.74 – Ação ao se clicar no botão MODO ALEATÓRIO

```
bt_modoAleatorio.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (nperg != 1) {
            AlertDialog.Builder adBuilder = new AlertDialog.Builder(Perfil.this);
            adBuilder.setTitle("Atenção");
            adBuilder.setMessage("Voce deseja continuar o jogo anterior?");
            adBuilder.setCancelable(true);
            adBuilder.setPositiveButton("Sim", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    Intent intent = new Intent(Perfil.this, Quiz.class);
                    intent.putExtra("codigo", codigo);
                    intent.putExtra("modo", 2);
                    startActivity(intent);
                    finish();
                }
            });
            adBuilder.setNegativeButton("Não (Novo Jogo)", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    CRUD.alterarRegistro(codigo, null, null, null, null,null, null, "1", "0", "0");
                    Intent intent = new Intent(Perfil.this, Quiz.class);
                    intent.putExtra("codigo", codigo);
                    intent.putExtra("modo", 2);
                    startActivity(intent);
                    finish();
                }
            });
            VerificaJogo = adBuilder.create();
            VerificaJogo.show();
        } else {
            Intent intent = new Intent(Perfil.this, Quiz.class);
            intent.putExtra("codigo", codigo);
            intent.putExtra("modo", 2);
            startActivity(intent);
            finish();
        }
    }
});
```

Foram implementadas as ações para mais dois botões na classe *Perfil*, a correspondente ao botão *Pontuações* que ao ser clicado, abrirá a tela *Pontuações* passando como parâmetro das trocas de tela o *id* do usuário escolhido para realizar o *quiz*, e a ação correspondente ao clique no botão *Voltar* que simplesmente finalizará a tela Perfil. A implementação das ações para esses botões pode ser vista na Figura 4.75.

Figura 4.75 – Ações dos botões Pontuações e Voltar

```
bt_pontuacoes.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        Intent intent = new Intent(Perfil.this, Pontuacoes.class);
        intent.putExtra("codigo", codigo);
        startActivity(intent);


    }
});

bt_voltar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        finish();
    }
});
```

4.5.11 Tela Pontuações

A tela *Pontuações* irá apresentar ao utilizador da aplicação o nome do usuário escolhido para jogar o *quiz*, as porcentagens de acertos obtidas na realização anterior, tanto no modo campanha como no modo aleatório, e um botão voltar a tela Perfil. Vale frisar que nas pontuações do modo campanha terá além da pontuação total do modo, o percentual de cada módulo do mesmo

Figura 4.76 – Tela Pontuações

4.5.11.1 Arquivo activity_pontuacoes.XML

Para *layout* da tela *Pontuações* foram definidos como parâmetros *latyout_width*, *layout_height* e *orientation*. Um dos componentes desse *layout*, como pode ser visto na Figura 4.77, é um *TextView* destinado a receber o nome do usuário.

Figura 4.77 – Componente TextView do arquivo activity_pontuacoes.XML

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/nome"
        android:layout_gravity="center"
        android:textSize="20dp"
        android:textStyle="bold"
        android:padding="16dp"
    />
```

Com intuito de uma melhor organização e distribuição dos textos destinados a exibir as pontuações foram criados vários componentes do tipo *layout*. Um desses *layouts*, como pode ser visto na Figura 4.78, tem como componentes *TextView's* com os nome dos módulos do modo campanha, e um dos parâmetros desse *TextView* é o *textAppearance* que permite dependendo do valor recebido alterar a aparência e o estilo da fonte do texto.

Figura 4.78 – Utilizando o método textAppearance

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_weight="2">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="Ângulos e Arcos: "
        android:id="@+id/textView1" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="Funções e relações trigonométrica: "
        android:id="@+id/textView2" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="Triângulos: "
        android:id="@+id/textView3" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="Total: "
        android:id="@+id/textViewT" />

</LinearLayout>
```

Outro *layout* usado como componente possui componentes *TextView's* destinados a mostrar o valor das porcentagens da realização de cada módulo e do total do modo campanha, como pode ser visto na Figura 4.79.

Figura 4.79 – TextView's para mostrar as pontuações dos módulos

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">

     <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:id="@+id/textM1"
        android:text="0"/>

     <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:id="@+id/textM2"
        android:text="0" />

     <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:id="@+id/textM3"
        android:text="0" />

     <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:id="@+id/textMT"
        android:text="0" />

</LinearLayout>
```

Para a pontuação do modo aleatório um *LinearLayout* com orientação horizontal foi usado, tendo como componentes três *TextView*'s, um com o texto "Pontuações", outro destinado a receber o valor da pontuação adquirida até o momento que o usuário realizará este modo de jogo e outro com o símbolo de porcentagem (%), como pode ser melhor visualizado na Figura 4.80.

Figura 4.80 – TextView's para mostrar as pontuações do modo aleatório

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="Pontuação: "
        android:layout_marginLeft="10dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:id="@+id/textpontale"
        android:text="0" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text=" %" />

</LinearLayout>
```

Outro componente do *layout* principal é um botão, destinado a voltar a tela anterior.

4.5.11.2 Classe Pontuacoes.java

Inicialmente na classe *Pontuacoes* foram criados os objetos para manipulação do banco de dados, manipulação de um botão, manipulação dos *TextView*'s, um inteiro e um cursor. No método *setContentView* a referência do arquivo *activity_pontuacoes* foi passada. Logo após ocorreu a vinculação dos objetos criados com os componentes visuais a qual se deseja manipular e instanciou o *CRUD* como um objeto da classe *ManipulaBanco*, como pode ser visto na Figura 4.81.

Figura 4.81 – Criando e inicializando os objetos da classe Pontuações

```
ManipulaBanco CRUD;
Button Bt_voltar;
TextView text_nome, text_mod1, text_mod2, text_mod3, text_total, text_totala;
int codigo;
Cursor Cursor;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_pontuacoes);

    Bt_voltar = (Button) findViewById(R.id.voltar);
    text_nome = (TextView) findViewById(R.id.nome);
    text_mod1 = (TextView) findViewById(R.id.textM1);
    text_mod2 = (TextView) findViewById(R.id.textM2);
    text_mod3 = (TextView) findViewById(R.id.textM3);
    text_total = (TextView) findViewById(R.id.textMT);
    text_totala = (TextView) findViewById(R.id.textpontale);
```

O inteiro criado recebe o valor passado pela troca de telas, o cursor recebe o retorno da função *carregaDadoById* passado como o parâmetro esse inteiro. Os *TextView*'s recebem os valores do nome, das pontuações totais dos dois modos de jogo e as pontuações de cada módulo do modo campanha, como pode ser visto na Figura 4.82. A ação para o botão Voltar foi definida como simplesmente finalizar a tela *Pontuações*.

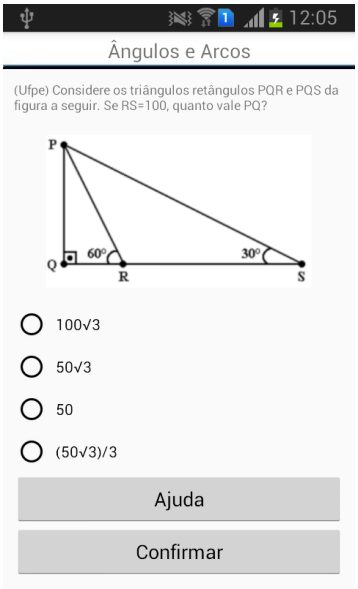
Figura 4.82 – Aplicação de Matemática Financeira

```
codigo = this.getIntent().getIntExtra("codigo" , 0);
Cursor = CRUD.carregarDadoById(codigo);

text_nome.setText(Cursor.getString(Cursor.getColumnIndexOrThrow(CriarBanco.NOME)));
text_mod1.setText(Cursor.getString(Cursor.getColumnIndexOrThrow(CriarBanco.MOD1)));
text_mod2.setText(Cursor.getString(Cursor.getColumnIndexOrThrow(CriarBanco.MOD2)));
text_mod3.setText(Cursor.getString(Cursor.getColumnIndexOrThrow(CriarBanco.MOD3)));
text_total.setText(Cursor.getString(Cursor.getColumnIndexOrThrow(CriarBanco.TOTAL)));
text_totala.setText(Cursor.getString(Cursor.getColumnIndexOrThrow(CriarBanco.TOTALA)));

Bt_voltar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        finish();
    }
});
```

4.5.12 Tela Quiz

A tela *Quiz* será onde o utilizador da aplicação encontrará as perguntas e o grupo de respostas, onde deverá ser escolhida somente uma. A tela é composta pelo nome do módulo a qual a pergunta atual se refere, o texto referente a mesma, uma imagem complementar da pergunta caso necessário, quatro possíveis respostas, um botão *Ajuda* que leva às telas para auxiliar na resolução da pergunta e o botão *Confirmar* que efetiva a resposta escolhida pelo jogador.

Figura 4.83 – Tela Quiz



4.5.12.1 Arquivo activity_quiz.XML

Para estruturação do *layout* da tela *Quiz* o gerenciador de *layout LinearLayout* com os parâmetros *layout_width*, *layout_height* e *orientation*. Os componentes desse *layout* principal foram dois *layouts* e dois botões. O primeiro componente *layout* possui como um dos seus parâmetros o *android:background*, esse parâmetro permite modificar o fundo do *layout* ou de algum componente de *layout*, podem ser passados como valor uma referência de uma imagem, como pode ser visto na Figura 4.84, sendo assim a imagem tomará o fundo desses componentes ou a referência de uma cor para deixar o fundo com a tonalidade escolhida. Esse componente *layout* possui um componente *TextView* destinado a receber o tema da disciplina a qual a pergunta pertence.

Figura 4.84 – Componente layout do *arquivo activity_quiz.XML*

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@drawable/cabecalho">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/nmmd"
            android:gravity="center"
            android:textSize="18sp"/>

    </LinearLayout>
```

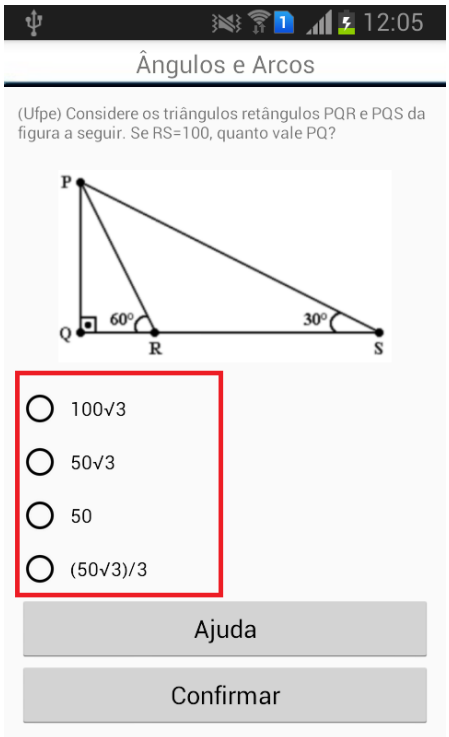
O outro componente *layout* possui como componente um *TextView* destinado para o texto das perguntas, um dos seus parâmetros é o *android:textColor*, esse parâmetro permite alterar a cor da fonte do texto, sendo necessário passar a referência da cor desejada. Como pode ser visto na Figura 4.85, para esse parâmetro foi passado a referência da cor preta “#000000”. Outro componente de *layout* foi um *ImageView* destinado a receber a imagem complementar caso a pergunta precise.

Figura 4.85 – Aplicação de Matemática Financeira

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="16dp"
        android:textColor="#000000"/>

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imagepg"
        android:layout_gravity="center"
        android:layout_weight="1"/>
```

Também é componente desse *layout* um *RadioGroup* com 4 *RadioButton's*, para o melhor entendimento do que é o *RadioGroup* com os *RadioButonI, a* Figura 4.86 ilustra-os os destacando pelo retângulo.

Figura 4.86 – RadioGroup



O *RadioGroup* será um grupo de opções (respostas) selecionáveis pelo usuário, onde só será possível selecionar uma delas, casa o utilizador da aplicação

marque uma opção e depois tente marcar outra, esta será selecionada e a marcada anteriormente será desconsiderada. Essas opções são os *RadioButton* destinados a receber os textos das possíveis respostas para o usuário escolher. A implementação visual para o *RadioGroup* e seus *RadioButton's* pode ser vista na Figura 4.87.

Figura 4.87 – Criando o RadioGroup com seus respectivos RadioButton's

```
<RadioGroup
    android:id="@+id/radioGroup1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="14dp">

    <RadioButton
        android:id="@+id/radio0"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="RadioButton"
        android:textSize="16dp"/>

    <RadioButton
        android:id="@+id/radio1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="RadioButton"
        android:textSize="16dp"/>

    <RadioButton
        android:id="@+id/radio2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="RadioButton"
        android:textSize="16dp"/>

    <RadioButton
        android:id="@+id/radio3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="RadioButton"
        android:textSize="16dp"/>
</RadioGroup>
```

Os botões do *layout* principal, que a declaração de seus parâmetros pode ser vista na Figura 4.88, são destinados ao botão *Ajuda* e ao botão *Confirmar*.

Figura 4.88 – Declaração dos botões do *layout* principal

```
<Button
    android:id="@+id/ajuda"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Ajuda"/>

<Button
    android:id="@+id/buttonConfirma"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Confirmar" />
```

4.5.12.2 Classe Quiz.java

No início da classe *Quiz* são declaradas as variáveis/objetos que serão utilizadas pela classe, como pode ser visto na Figura 4.89

Figura 4.89 – Criando os objetos e as variáveis utilizadas pela classe Quiz

```
Questao QuestaoAtual = new Questao();
ManipulaBanco CRUD;
Cursor cursor;
TextView txtQuestao, nmmd;
ImageView imagemperg;
RadioGroup rGroup;
RadioButton optA, optB, optC, optD, resposta;
Button btConfirma, btAjuda;
AlertDialog Concluido, alertDialog, avaliacao;
int pontuacaoTemp = 0, percTemp, nques = 1, nmodulo, mod1,
        mod2, mod3, T, codigo, modo, nperg, totala, acert;
String img;
List<Questao> quesList;
```

As variáveis e os objetos são o *CRUD* que é um objeto da classe *ManipulaBanco*, os objetos para manipular os componentes do arquivo XML que essa classe utilizará, os objetos para trabalhar com *AlertDialog*, o *cursor* que será um objeto da classe *Cursor*, os inteiros para trabalhar com as perguntas e as pontuações, uma *string* para referenciar a imagem para o auxílio do entendimento da pergunta, um objeto da classe *List* e outro objeto do tipo *Questao.*

O objeto da classe List, terá como coleção elementos do tipo *Questao*, por ter sido criado como *List<Questao>*.

O objeto do tipo *Questao* é inicializado chamando o construtor sem parâmetros, como explicado na seção 4.4.2.

Algumas da variáveis criadas e suas devidas funções são:

- **pontuacaoTemp:** criado para armazenar os acertos das perguntas referentes ao módulo em jogo;
- **percTemp:** criado para armazenar a percentagem referente ao acertos das perguntas do módulo em jogo;
- **nques:** criado para determinar a quantidade de perguntas já respondida do módulo em jogo. É inicializada com o valor 1, justamente para ser referente a primeira pergunta a ser respondida. Seu valor será

incrementado até 5, ou seja, até a quinta pergunta para o módulo em jogo;

- **nmodulo:** criado para referenciar o módulo que deve ser buscado as perguntas.
- **mod1:** criado para referenciar a pontuação salva adquirida quando foi realizado o módulo 1;
- **mod2:** criado para referenciar a pontuação salva adquirida quando foi realizado o módulo 2;
- **mod3:** criado para referenciar a pontuação salva adquirida quando foi realizado o módulo 3;
- **T:** criado para referenciar a pontuação salva adquirida quando foi realizado todos os módulos, ou seja, a pontuação final;
- **codigo:** criado para receber o valor do *id* do usuário, passado por parâmetro na troca de telas;
- **modo:** criado para receber o valor referente ao modo de jogo escolhido, passado por parâmetro na troca de telas;
- **nperg:** criado para receber o valor do números de perguntas respondidas no modo aleatório. Caso sair do *quiz* sem completar 15 perguntas, o utilizador da aplicação terá direito de escolher continuar o modo aleatório até completar este número de perguntas ou começar um novo jogo zerando o número de perguntas respondidas;
- **totala:** criado para receber a percentagem de perguntas acertadas entre as 15 a se responder para completar o modo aleatório;
- **acert:** criado para receber o valor de perguntas acertadas entre as 15 a se responder para completar o modo aleatório;
- **QuestaoAtual:** criado para receber os valores necessários para trabalhar com a atual questão que deverá ser respondida.

Pelo método *setContentView* o arquivo *activiy_quiz* foi referenciado para a parte visual da tela. Logo após os objetos criados na classe foram ligados aos componentes do arquivo XML que irão corresponder. O objeto da classe *List* foi inicializado como um *ArrayList*, que é uma implementação da *List* auxiliado por matriz, onde são suportados as operações de adicionar, remover e substituir, aceitando todos

os tipos de elementos incluindo o *null*. Também foi inicializado o objeto da classe *ManipulaBanco*, como pode ser visto na Figura 4.90.

Figura 4.90 – Inicializando os objetos e variáveis da classe Quiz

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_quiz);

    txtQuestao = (TextView)findViewById(R.id.textView1);
    optA = (RadioButton)findViewById(R.id.radio0);
    optB = (RadioButton)findViewById(R.id.radio1);
    optC = (RadioButton)findViewById(R.id.radio2);
    optD = (RadioButton)findViewById(R.id.radio3);
    btConfirma = (Button)findViewById(R.id.buttonConfirma);
    btAjuda = (Button) findViewById(R.id.ajuda);
    imagemperg = (ImageView) findViewById(R.id.imagepg);
    nmmd = (TextView) findViewById(R.id.nmmd);

    quesList = new ArrayList<>();
    CRUD = new ManipulaBanco(getBaseContext());
```

Como pode ser visto na Figura 4.91, o inteiro *codigo* recebe o valor referente ao *id* do usuário e o inteiro *modo* recebe o valor referente ao modo de jogo que foi escolhido pelo utilizador da aplicação na tela anterior, ambos valores foram passados por parâmetros na troca de telas. A função *addQuestao* é chamada e o objeto da classe *Cursor* recebe o retorno da função *carregarDadoById* passando como parâmetro o inteiro *codigo*.

Figura 4.91 – Recebendo os valores provindos da troca de telas

```
codigo = this.getIntent().getIntExtra("codigo", 0);
modo =  this.getIntent().getIntExtra("modo", 0);
CRUD.addQuestao();
cursor = CRUD.carregarDadoById(codigo);
```

Após esses procedimentos, estruturas condicionais (*if*) são declaradas, cada uma terá um procedimento dependendo do modo do jogo escolhido, campanha ou aleatório. Para um melhor entendimento os procedimentos destes dois módulos, três das funções utilizadas, o método *onClick* do botão *Ajuda* e ação quando acionada a tecla *retornar* do dispositivo móvel serão explicados separadamente em tópicos.

4.5.12.2.1 Função setQuestView

A função *setQuestaoView*, a qual sua declaração pode ser vista na Figura 4.92, atribui as variáveis referente ao texto da pergunta (txtQuestao) e aos *radiobutton's* (optA, optB, optC e optD) com, respectivamente, os valores do texto da pergunta e os textos das respostas possíveis da questão referente a *QuestaoAtual*.

Um inteiro *mod* é criado e recebe o valor referente ao módulo que a pergunta atual corresponde, um *switch* é utilizado para atribuir ao um dos *TextView´s* do *layout* o nome do módulo o qual o valor *mod* referencia.

A variável *img* é criada e inicializada com a referência, ou seja, o nome, da imagem da questão atual e caso seu valor seja diferente de *null*, ou seja, a pergunta faz necessário uma imagem para sua complementação, o componente do tipo *ImageView* do *layout* receberá, através do método *setImagemDrawable*, tal imagem. Para a identificação da imagem a qual a *ImagemView* receberá foi usado o método *getResources().getDrawable(getResources().getIdentifier())*, que quando passado como parâmetros o nome da imagem e o nome da pasta em que se encontra tal imagem, será busca com tais informações. Caso contrário a variável *imagemperg*, criada inicialmente na classe *Quiz*, receberá o valor *null*, consequentemente o componente *ImagemView* criado não receberá imagem.

Figura 4.92 – Função setQuestãoView

```
private void setQuestaoView() {
    txtQuestao.setText(QuestaoAtual.getQUESTAO());
    optA.setText(QuestaoAtual.getOPTA());
    optB.setText(QuestaoAtual.getOPTB());
    optC.setText(QuestaoAtual.getOPTC());
    optD.setText(QuestaoAtual.getOPTD());

    int mod;
    mod = QuestaoAtual.getQMOD();
    switch (mod){
        case 1: nmmd.setText("Ângulos e Arcos");
            break;

        case 2: nmmd.setText("Funções e relações trigonométricas");
            break;

        case 3: nmmd.setText("Triângulos");
            break;
    }

    img = QuestaoAtual.getIMG();
    if (img != null) {

        Drawable drawable = getResources().getDrawable(getResources().
                getIdentifier(img, "drawable", getPackageName()));
        imagemperg.setImageDrawable(drawable);

    } else{

        imagemperg.setImageDrawable(null);
    }

}
```

4.5.12.2.2 Funçãofuncionamentomodo1

A função *funcionamentomodo1* será usada como um dos procedimentos caso for escolhido o modo de jogo campanha. A função inicia com uma estrutura condicional (*if*), como pode ser visto na Figura 4.93, caso a variável *nques* possuir valor 5, ou seja, for a quinta pergunta respondida de um mesmo módulo, essa mesma variável receberá o valor 0 para iniciar uma nova sequência de perguntas a responder. A variável *perctemp* receberá o resultado do cálculo de *pontuacaoTemp* multiplicado por 100 e depois dividido por 5, ou seja, da percentagem de perguntas acertadas dentre as 5 respondidas de um mesmo módulo.

A mesma estrutura condicional (*if*) tem um *switch*, onde dependendo do valor da variável *nmodulo*, será resgatado do banco de dados as pontuações dos outros módulos para ser efetuado o cálculo da percentagem total das perguntas acertadas nos três módulos e atribuindo o resultado deste cálculo a variável *T*. Logo após, através da função *alteraRegistro*, as novas percentagens calculadas do módulo atual e do total são registradas no banco de dados, referente ao usuário em questão. Também é alterado no banco de dados, através dessa função, o valor do módulo que o usuário irá interagir, ou seja, a cada módulo que o usuário completa é alterado no banco de dados para o próximo módulo de qual deverá ser apresentado as próximas perguntas.

Saindo do *switch* a variável *nmodulo* é incrementada de 1, ou seja, o usuário avançou de módulo, como pode ser visto na figura 4.94. Uma nova estrutura condicional (*if*) é declarada, caso o valor da variável *nmodulo* for diferente de 4, ou seja, o usuário ainda não realizou os 3 módulos existentes, a variável *questList* receberá uma nova lista de questões referente ao módulo atual e essas questões são embaralhadas.

Se o *nques* for igual a 5 representará o término de um módulo, uma caixa de diálogo será apresentada, com a mensagem “Etapa concluída” e com a percentagem de acertos adquirida ao completar o módulo. Quando o utilizador da aplicação clicar no botão *OK* da caixa de diálogo, será feita a verificação se já foi completado os 3 módulos, para que caso for completado a tela *Resultado* possa ser chamada.

Para finalizar os procedimentos dessa estrutura condicional (*if*), a variável *pontuacaoTemp* recebe o valor 0 e a caixa de diálogo é criada e apresentada ao utilizador do sistema.

Figura 4.93 – Função funcionamento

```
private void funcionamentomodo1(){


    if (nques == 5) {
        nques = 0;
        percTemp = (pontuacaoTemp * 100) / 5;

        switch (nmodulo) {
            case 1:
                mod2 = cursor.getInt(cursor.getColumnIndexOrThrow(CriarBanco.MOD2));
                mod3 = cursor.getInt(cursor.getColumnIndexOrThrow(CriarBanco.MOD3));
                T = (percTemp + mod2 + mod3) / 3;
                CRUD.alterarRegistro(codigo, null, "2", Integer.toString(percTemp),
                        null, null, Integer.toString(T), null, null, null);
                break;

            case 2:
                mod1 = cursor.getInt(cursor.getColumnIndexOrThrow(CriarBanco.MOD1));
                mod3 = cursor.getInt(cursor.getColumnIndexOrThrow(CriarBanco.MOD3));
                T = (mod1 + percTemp + mod3) / 3;
                CRUD.alterarRegistro(codigo, null, "3", null, Integer.toString(percTemp),
                        null, Integer.toString(T), null, null, null);
                break;

            case 3:
                mod1 = cursor.getInt(cursor.getColumnIndexOrThrow(CriarBanco.MOD1));
                mod2 = cursor.getInt(cursor.getColumnIndexOrThrow(CriarBanco.MOD2));
                T = (mod1 + mod2 + percTemp) / 3;
                CRUD.alterarRegistro(codigo, null, "1", null, null,
                        Integer.toString(percTemp), Integer.toString(T), null, null, null);
                break;
        }
```

Figura 4.94 – Apresentando a caixa de diálogo de um término de um módulo

```
            nmodulo++;
            if (nmodulo != 4) {
                quesList = CRUD.getQuestoes(nmodulo);
                if (quesList != null && quesList.size() != 0) {
                    Collections.shuffle(quesList);
                }
            }


            AlertDialog.Builder builder = new AlertDialog.Builder(Quiz.this);
            builder.setCancelable(false);
            builder.setTitle("Etapa concluída");
            builder.setMessage("Acertos: " + percTemp + "%");

            builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    Concluido.dismiss();
                    if (nmodulo == 4) {
                        Intent intentR = new Intent(Quiz.this, Resultado.class);
                        intentR.putExtra("codigo", codigo);
                        intentR.putExtra("modo", modo);
                        startActivity(intentR);
                        finish();
                    }
                }
            });

            pontuacaoTemp = 0;

            Concluido = builder.create();
            Concluido.show();
```

Caso *nques* não for igual a 5 e consequentemente não entrar na primeira estrutura condicional (*if*) da função *funcionamentomodo1*, será verificado se o *nmodulo* é diferente de 4, a *questList* não está vazia e seu tamanho é diferente de 0, caso satisfaça todas essas condições, *nques* é incrementado de 1, *QuestaoAtual* recebe a questão da lista referente ao índice do valor de *nques* e a função *setQuestaoView* é chamada, como pode ser visto na figura 4.95.

Ao final da função *funcionamentomodo1*, através do método *clearCheck*, é desmarcada a opção escolhida como resposta da questão anterior.

Figura 4.95 – Usando o método clearCheck

```
if (nmodulo != 4 && quesList != null && quesList.size() != 0) {
    nques++;
    QuestaoAtual = quesList.get(nques);
    setQuestaoView();
}

rGroup.clearCheck();
```

4.5.12.2.3 Função funcionamentomodo2

Na implementação do funcionamento do *quiz* caso for escolhido o modo de jogo aleatório a função *funcionamentomodo2* será chamada. Nesta função são encontrada estruturas condicionais (*if*). A primeira é caso *nperg* for igual a 15, ou seja, o usuário respondeu 15 perguntas, a variável *totala* recebe o resultado do cálculo de percentagem das questão acertadas, através da função *alterarRegistro* os campos referentes ao número de perguntas recebe o valor 1 indicando que o usuário iniciará um novo jogo tendo que responder sua primeira pergunta e o campo referente ao número de acertos recebe o valor 0 e a classe referente a tela *Resultado* é chamada como pode ser visto na figura 4.96.

A segunda estrutura condicional (*if*), que também pode ser visualizada na figura 4.96, é caso *nperg* for diferente de 15 ou seja, o usuário ainda não respondeu às 15 perguntas, e *quesList* for diferente de vazio e seu tamanho for diferente de 0, *nperg* será incrementa de 1, *QuestaoAtual* receberá uma nova questão da *questList* com índice do valor de *nperg*, a variável *totala* receberá o resultado do cálculo da percentagem das perguntas acertadas entre as 15 a se responder, através da função

alterarRegistro o número da pergunta a se responder e a percentagem total de acertos é salvo no banco de dados e a classe referente a tela *Resultado* é chamada.

No final da função *funcionamentomodo2*, é utilizado o método *clearCheck* novamente.

Figura 4.96 – Chamando a tela Resultado

```
private void funcionamentomodo2() {

    if (nperg == 15) {

        totala = (acert * 100) / 15;
        CRUD.alterarRegistro(codigo, null, null, null, null, null, null,
                "1", "0", Integer.toString(totala));
        Intent intentR = new Intent(Quiz.this, Resultado.class);
        intentR.putExtra("codigo", codigo);
        intentR.putExtra("modo", modo);
        startActivity(intentR);
        finish();

    }

    if (nperg != 15 && quesList != null && quesList.size() != 0) {
        nperg++;
        QuestaoAtual = quesList.get(nperg);

        totala = (acert * 100) / 15;
        CRUD.alterarRegistro(codigo, null, null, null, null,
                    null, null, Integer.toString(nperg),
                Integer.toString(acert), Integer.toString(totala));

        setQuestaoView();
    }

    rGroup.clearCheck();
}
```

4.5.12.2.4 Modo Campanha

Caso o valor do inteiro *modo* seja igual a 1, ou seja, o modo campanha foi escolhido pelo utilizador do sistema, o inteiro *nmodulo* receberá o valor do módulo que será necessário buscar as questões e a variável *quesList* receberá, através da função getQuestoes, as questões referentes ao módulo que o valor foi passado como parâmetro.

Posteriormente, virá uma outra estrutura condicional (*if*) onde, de forma essencial, verificará se a lista *quesList* não está vazia e seu tamanho é diferente de 0 para que ocorra, através do método *Collections.shuffle* o emparelhamento de seus elementos, ou seja, das questões na lista. E a variável *QuestaoAtual* receberá a

questão na posição do valor de *nques*, como pode ser visto na Figura 4.97. Logo após, a função *setQuestaoView* é chamada.

Figura 4.97 – Usando o método Collections.shuffle

```
if (modo == 1) {

    nmodulo = cursor.getInt(cursor.getColumnIndexOrThrow(CriarBanco.MODULO));
    quesList = CRUD.getQuestoes(nmodulo);

    if (quesList != null && quesList.size() != 0) {
        Collections.shuffle(quesList);
        QuestaoAtual = quesList.get(nques);

    }

    setQuestaoView();
```

Após chamada à função *setQuestView* foi estipulado as ações a serem tomadas para caso o botão *Confirmar* presente nesse *layout* for clicado. Tais ações são:

- A variáveis criadas *rGroup* e *resposta* são vinculadas, respectivamente, aos componentes *RadioGroup* e *RadioButton* que foi selecionado pelo usuário da aplicação;
- Se o valor da variável *resposta* for diferente de *null*, ou seja, o usuário marcou alguma opção, e caso essa opção for a resposta correta (tal verificação é feita usando o método *equals* entre o texto contido no campo *RESPOSTA* referente a pergunta atual no banco de dados e a variável *resposta*, caso forem iguais é retornado o valor "*TRUE*", caso contrário retorna "*FALSE*") a variável *pontuacaoTemp* é incrementada de 1, é criado uma caixa de diálogo com o título "PARABÉNS" e a mensagem "Sua resposta está correta!" e caso o botão *OK* da caixa de diálogo for acionado a função *funcionamentomodo1* é chamada, como pode ser visto na Figura 4.98;
- Se o valor da variável *resposta* for diferente de *null*, ou seja, o usuário marcou alguma opção, e caso essa opção for uma resposta incorreta uma caixa de diálogo é criada e mostrada para o usuário com o título "QUE PENA!" e a mensagem "Sua reposta está errada!" e quando clicado no botão *OK* da caixa de diálogo a função *funcionamentomodo2* é chamada, como pode ser visto na Figura 4.99;

Figura 4.98 –Chamando a função funcionamentomodo1

```
btConfirma.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        rGroup = (RadioGroup) findViewById(R.id.radioGroup1);
        resposta = (RadioButton) findViewById(rGroup.getCheckedRadioButtonId());
        if (resposta != null) {
            if (QuestaoAtual.getRESPOSTA().equals(resposta.getText())) {
                pontuacaoTemp++;

                AlertDialog.Builder builder = new AlertDialog.Builder(Quiz.this);
                builder.setCancelable(false);
                builder.setTitle("PARABÉNS!");
                builder.setMessage("Sua resposta está correta!");


                builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        avaliacao.dismiss();

                        funcionamentomodo1();
                    }
                });

                avaliacao = builder.create();
                avaliacao.show();
```

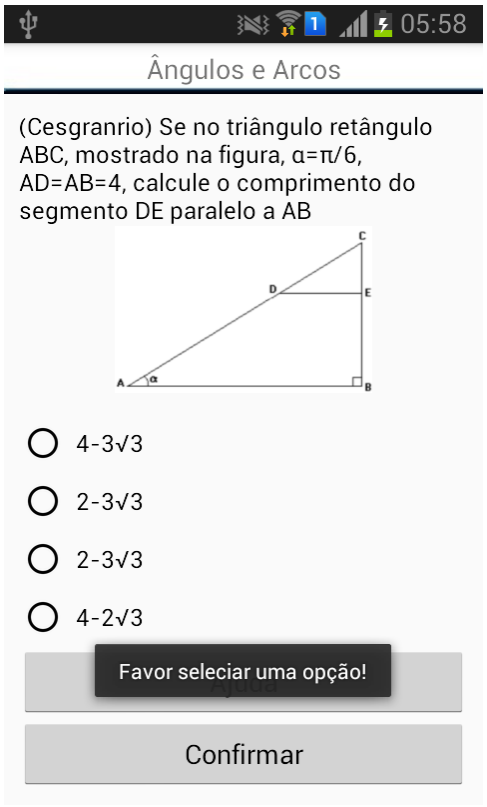
Figura 4.99 – Chamando a função funcionamentomodo2

```
    } else {
        AlertDialog.Builder builder = new AlertDialog.Builder(Quiz.this);
        builder.setCancelable(false);
        builder.setTitle("QUE PENA!");
        builder.setMessage("Sua resposta está incorreta!");

        builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                avaliacao.dismiss();

                funcionamentomodo1();

            }
        });

        avaliacao = builder.create();
        avaliacao.show();
    }

} else {

    Toast.makeText(Quiz.this, "Favor seleciar uma opção!",
            Toast.LENGTH_SHORT).show();

}
```

- Se não for selecionado nenhuma opção, ou seja, a variável *resposta* assumir valor *null*, uma mensagem será apresentada ao usuário o

alertando que deve optar e selecionar alguma reposta, como pode ser visto na Figura 4.100.

Figura 4.100 – Alerta de nenhuma resposta escolhida



4.5.12.2.5 Modo Aleatório

Caso o valor do inteiro *modo* seja igual a 2, ou seja, o modo aleatório foi escolhido pelo usuário, o inteiro *nperg* receberá o valor referente a qual pergunta o usuário deverá responder de acordo com o seu valor salvo no campo *NPERG* do banco de dados, por exemplo, se nperg for igual a 1 será a primeira pergunta a responder, se for 2 será segunda e assim sucessivamente. A variável *acert* receberá o valor contido no campo *ACERT* referente ao usuário no banco de dados e *questList* receberá, através da função *getTodasQuestoes*, todas as questões salva no banco de dados.

Logo após, uma estrutura condicional (*if*) onde, de forma necessária, verificará se a lista *quesList* não está vazia e seu tamanho é diferente de 0, para que ocorra, através do método *Collections.shuffle* o emparelhamento de seus elementos, ou seja,

de todas as questões da lista. E a variável *QuestaoAtual* receberá a questão na posição do valor de *nperg*, como pode ser visto na figura 4.101. Logo após, a função *setQuestaoView* é chamada.

Figura 4.101 – Aplicação de Matemática Financeira

```
if(modo == 2){

    nperg = cursor.getInt(cursor.getColumnIndexOrThrow(CriarBanco.NPERG));
    acert = cursor.getInt(cursor.getColumnIndexOrThrow(CriarBanco.ACERT));
    quesList = CRUD.getTodasQuestoes();


    if (quesList != null && quesList.size() != 0) {
        Collections.shuffle(quesList);
        QuestaoAtual = quesList.get(nperg);
    }

    setQuestaoView();
```

Após chamada à função *setQuestView* foi determinados os eventos a serem executados ao acionar o botão Confirmar presente nesse *layout*, e o modo aleatório tenha sido escolhido. Tais eventos são:

- A variáveis criadas *rGroup* e *resposta* são vinculadas, respectivamente, aos componentes *RadioGroup* e o *RadioButton* que foi selecionado pelo usuário da aplicação;
- Se o valor da variável *resposta* for diferente de *null* e caso opção escolhida como resposta for a correta, a variável *acert* é incrementada de 1 e é criado uma caixa de diálogo com o título “PARABÉNS” e com a mensagem “Sua resposta está correta!”. Caso o utilizador da aplicação clicar no botão *OK* da caixa de diálogo a função *funcionamentomodo2* é chamada, como pode ser visto na Figura 4.102;
- Se o valor da variável “*resposta*” for diferente de *null*, ou seja, o usuário marcou alguma opção, e caso essa opção for uma resposta incorreta uma caixa de diálogo é criada e mostrada para o usuário com o título “QUE PENA!” e com a mensagem “Sua reposta está errada!” e quando clicado no botão *OK* da caixa de diálogo a função *funcionamentomodo2* é chamada, como pode ser visto na figura 4.103;

Figura 4.102 – Chamando a função funcionamentomodo2

```
btConfirma.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        rGroup = (RadioGroup) findViewById(R.id.radioGroup1);
        resposta = (RadioButton) findViewById(rGroup.getCheckedRadioButtonId());
        if (resposta != null) {
            if (QuestaoAtual.getRESPOSTA().equals(resposta.getText())) {
                acert++;

                AlertDialog.Builder builder = new AlertDialog.Builder(Quiz.this);
                builder.setCancelable(false);
                builder.setTitle("PARABÉNS!");
                builder.setMessage("Sua resposta está correta!");

                builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        avaliacao.dismiss();

                        funcionamentomodo2();

                    }
                });

                avaliacao = builder.create();
                avaliacao.show();
```

Figura 4.103 – Caso a resposta escolhida for incorreta

```
            } else {

                AlertDialog.Builder builder = new AlertDialog.Builder(Quiz.this);
                builder.setCancelable(false);
                builder.setTitle("QUE PENA!");
                builder.setMessage("Sua resposta está errada!");

                builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        avaliacao.dismiss();

                        funcionamentomodo2();

                    }
                });

                avaliacao = builder.create();
                avaliacao.show();

            }

        } else {

            Toast.makeText(Quiz.this, "Favor seleciar uma opção!",
                    Toast.LENGTH_SHORT).show();

        }
```

- Se não for selecionado nenhuma opção, ou seja, a variável *resposta* assumir valor *null*, uma mensagem será apresentada ao usuário o alertando que deve optar e selecionar alguma reposta.

4.5.12.2.6 Botão Ajuda

No método *OnClick* do botão ajuda, que pode ser visto na Figura 4.104, o inteiro *mod* foi criado e foi inicializado com o valor referente ao módulo que a *QuestaoAtual* pertence, também foi implementado um *switch* onde dependendo do valor do módulo a tela de *Ajuda* referente a este módulo será chamada, por exemplo, caso *mod* for igual a 1 será chamada a tela de *Ajuda* referente ao modulo *Ângulos e Arcos*.

Figura 4.104 – Método onClick do botão Ajuda

```
btAjuda.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        int mod;
        mod = QuestaoAtual.getQMOD();

        switch (mod){
            case 1:
                Intent intent = new Intent(Quiz.this, Ajuda1.class);
                startActivity(intent);
                break;

            case 2:
                Intent intent2 = new Intent(Quiz.this, Ajuda2.class);
                startActivity(intent2);
                break;

            case 3:
                Intent intent3 = new Intent(Quiz.this, Ajuda3.class);
                startActivity(intent3);
                break;
        }

    }
});
```

4.5.12.2.7 Tecla Retornar

Quando acionada a tecla *retornar* do dispositivo o qual está rodando a aplicação, estando na tela *Quiz*, uma caixa de diálogo será criada e mostrada ao usuário da aplicação, como pode ser visto na Figura 4.105. Esta caixa de diálogo terá

o título “Atenção” e a mensagem “Deseja realmente sair?” Caso a opção *Sim* for escolhida a tela *Quiz* fechará e a classe referente a tela *Perfil* será chamada, caso contrário, optando pela opção *Não* nenhuma ação além do remate da caixa de diálogo será tomada, ou seja, continuará na tela *Quiz* com a mesma pergunta de antes de se clicar na tecla *retornar*.

Figura 4.105 – Caixa de diálogo para caso se deseja sair do quiz

```
@Override
public void onBackPressed() {
    AlertDialog.Builder Builder = new AlertDialog.Builder(Quiz.this);
    Builder.setTitle("Atenção");

    Builder.setMessage("Deseja realmente sair?");
    Builder.setCancelable(false);
    Builder.setPositiveButton("Sim", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            Intent intent = new Intent(Quiz.this, Perfil.class);
            intent.putExtra("codigo", codigo);
            startActivity(intent);
            finish();
        }
    });
    Builder.setNegativeButton("Não", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            dialog.cancel();
        }
    });

    alertDialog = Builder.create();
    alertDialog.show();
}
```

4.5.13 Ajuda

Ao clicar no botão *Ajuda* da tela *Quiz*, o usuário irá deparar com opções de diferentes temas relacionado ao módulo, onde, optando por uma das opções até chegar a ajuda desejada, ocorrerá uma filtragem na busca. Essa opções serão apresentadas como botões em diferentes telas, pois cada módulo possui temas específicos a eles. A ajuda será apresentada através de uma imagem e a tela que mostrará essa imagem é única para os diferentes tipos de módulos e seus procedimentos são implementados na classe *AjudaImagem*, o que mudará será somente o caminho (as escolhas das opções/tema) até chegar a tela da imagem.

As telas da ajuda de cada módulo tem opções relacionadas aos assuntos do mesmo. As opções tem como objetivo filtrar a ajuda oferecida. As estruturas para a criação destas telas são muito semelhantes, mudando apenas o nome dos botões,

sendo assim, as mesmas serão explicadas juntamente a seguir. A tela que receberá imagem referente a ajuda dos diferentes temas dos módulos, será explicado, separadamente, por último. Haverá a apresentação das telas e a representação de como foi desenvolvido os seus arquivos XML e suas classes.

4.5.13.1 Tela

A tela com as opções dos assuntos referentes aos diferentes módulos do jogo, apresentará ao usuário os botões relacionados as opções sobre os temas do módulo em questão e um botão para sair da ajuda. Para o módulo Ângulos e Arcos, como pode ser visto na Figura 4.106, a tela apresentará as opções: Medida de Arcos, Medida de Ângulo e Ciclo Trigonométrico.

Figura 4.106 – Ajuda do módulo Ângulos e Arcos

Para o módulo Funções e Relações Trigonométricas, como pode ser visto na Figura 4.107, a tela apresentará as opções: Funções Circulares e Relações Fundamentais.

Figura 4.107 – Tela da ajuda do módulo Funções e Relações Trigonométricas

Para o módulo Triângulos, como pode ser visto na Figura 4.108, a tela apresentará as opções: Triângulos Retângulos *e* Triângulos Quaisquer.

Figura 4.108 – Tela da ajuda do módulo Triângulos

4.5.13.2 Arquivo XML

Os arquivos XML para as telas que irão filtrar as ajudas dos diferentes módulos da aplicação, terão como gerenciador do *layout* o *LinearLayout* utilizando os parâmetros *layout_width*, *layout_height*, *padding* e *orientation*, como pode ser visto na Figura 4.109.

Figura 4.109 – Gerenciador de *layout* das telas de filtragem da ajuda

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="10dp"
    android:orientation="vertical">
```

Um dos componentes do *layout* principal, é um *layout* composto pelos componentes do tipo botão referentes a cada tema/opções dos módulos, como pode ser visto na Figura 4.110, que mostra os componentes para as opções do módulo Retângulos.

Figura 4.110 – Componentes do tipo botão da ajuda do módulo Retângulos

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:layout_weight="1">

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/bt1"
    android:text="Triângulos Retângulos"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/bt2"
    android:text="Triângulo quisquer"/>

</LinearLayout>
```

Outro componente do *layout* principal, comum em todos os arquivos XML dessas telas, é um botão referente a sair da tela atual, como pode ser visto na Figura 4.111.

Figura 4.111 – Botão para sair da tela atual

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/sair"
    android:text="Sair da Ajuda"/>
```

4.5.13.3 Classe Java

Para as classes *.java* das telas com as opções de filtragem de ajuda de cada módulo, foram criados, incialmente, os objetos para manipular os componentes do *layout* que irá ser utilizado pelas classes. Após isto, a referência do arquivo de *layout* é passado para o método *setContentView* e os objetos criados são vinculados aos componentes desse arquivo. Como pode ser visto a Figura 4.112, o início da classe com os procedimentos referente ao modulo Ângulos e Arcos.

Figura 4.112 – Criando e inicializando os objetos e as variáveis

```
Button bt_1, bt_2, bt_3, sairaj;
int bt;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_ajuda1);

    bt_1 = (Button) findViewById(R.id.bt1);
    bt_2 = (Button) findViewById(R.id.bt2);
    bt_3 = (Button) findViewById(R.id.bt3);
    sairaj = (Button) findViewById(R.id.sair);
```

Nos métodos *onClick* dos botões das telas é chamada a classe referente a tela que apresentará imagem da ajuda e é passado o inteiro *bt* como parâmetro no método *putExtra* inicializado com o valor referente ao tema escolhido. O uso desse inteiro se dá ao fato de que, como a tela que apresentará a imagem com a ajuda ser única para diferentes tipos de tema, uma estrutura condicional (*if*), com esse inteiro, definirá qual imagem a ser apresentada na tela de acordo com o tema escolhido. O valor escolhido para o inteiro relacionado ao tema/opção que ele irá representa foi de forma crescente de acordo com a ordem em que os botões para os(as) temas/opções foram

implementados.

Já a ação para o acionamento do botão *Sair da Ajuda* é simplesmente finalizar a tela. Como exemplo, a Figura 4.113, mostra a implementação das ações ao acionar os botões da tela de ajuda referente ao módulo Funções e Relações Trigonométricas.

Figura 4.113 – Método *onClick* relacionado ao botões da tela de ajuda do módulo Funções e Relações Trigonométricas

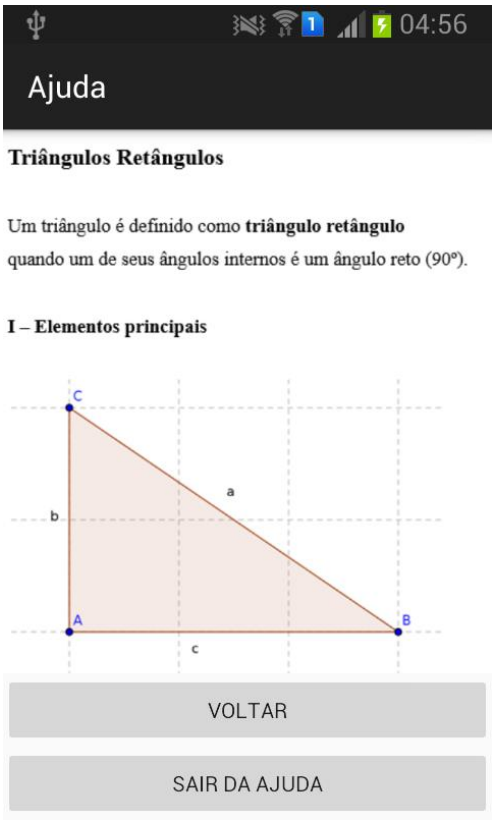
```
bt_1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(Ajuda2.this, AjudaImagem.class);
        intent.putExtra("bt", bt = 4);
        startActivity(intent);
        finish();
    }
});

bt_2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(Ajuda2.this, AjudaImagem.class);
        intent.putExtra("bt", bt = 5);
        startActivity(intent);
        finish();
    }
});

sairaj.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        finish();
    }
});
```

4.5.13.4 Imagem com ajuda

4.5.13.4.1 Tela

Esta tela será única para todos as opções de ajuda, ou seja, será só 1 arquivo XML e 1 classe java, para as opções escolhidas até chegar nela e apresentará 1 imagem, está imagem será diferente mediantes as opções escolhidas. Um exemplo de uma imagem com ajuda sendo apresentado por essa tela pode ser visto na Figura 4.114. A tela também apresentará um botão *Voltar* caso o usuário desejar retornar a tela com os temas e o botão *Sair da Ajuda* caso o usuário deseje retornar à pergunta a ser respondida.

Figura 4.114 – Exemplo de uma imagem com ajudas



4.5.13.4.2 Arquivo activity_ajuda_imagem.XML

O arquivo *acitvity_ajuda_imagem* será a representação visual da tela que apresentará a imagem com os auxílios para a resolução da pergunta a ser respondida. Como gerenciador do *layout*, foi utilizado o *LinearLayout* utilizando os parâmetros *layout_width*, *layout_height*, *padding* e *orientation*. Um dos componentes de *layout* principal é componente do tipo *ScrollView*, como pode ser visto na Figura 4.115.

Figura 4.115 – Componente ScrollView

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1">

        <ImageView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/imageaj"/>

    </ScrollView>
```

O componente *ScrollView* permitirá a rolagem da tela do dispositivo caso os componentes que o compõe ultrapasse o limite físico da tela, Neste caso o componente *ScrowView* possui um componente *ImageView*, caso a imagem referente ao *ImagemView* ter um tamanho maior que a tela pode suportar, ela não será cortada, e sim, haverá a possibilidade de rolar a tela para a visualização completa desta imagem.

Outro dois componentes desse *layout* principal, como pode ser visto na Figura 4.116, são 2 botões referente a voltar a tela anterior e a sair da ajuda.

Figura 4.116 – Outros componentes do arquivo activity_ajuda_imagem

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/voltar"
    android:text="Voltar"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/sair"
    android:text="sair da ajuda"/>
```

4.5.13.4.3 Classe Ajudaimagem.java

Inicialmente, nesta classe, foram criados os objetos para manipular os

componentes, 2 botões e 1 imagem, do *layout* que irá ser utilizado e o inteiro *bt*. Após isto, a referência do arquivo *activity_ajuda_imagem* é passado para o método *setContentView*, os objetos criados são vinculados aos componentes desse arquivo e o inteiro *bt* receberá o valor passado como parâmetro na troca de telas referente a opção escolhida nas telas anteriores a esta. Como pode ser visto na Figura 4.117.

Figura 4.117 – Criando objetos e variáveis da classe Ajudaimagem

```
ImageView ajuda;
Button btvoltar, btsairaj;
int bt;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_ajuda_imagem);

    ajuda = (ImageView) findViewById(R.id.imageaj);
    btvoltar = (Button) findViewById(R.id.voltar);
    btsairaj = (Button) findViewById(R.id.sair);

    bt = this.getIntent().getIntExtra("bt", 0);
```

Logo após, um *switch* é implementado para que de acordo com o valor do *bt*, ou seja, de acordo com a opção escolhida para chegar a esta tela, uma imagem com a ajuda para a pergunta será mostrada pelo *ImageView* do *layout*. A implementação de tal *switch* pode ser vista na Figura 4.118.

Figura 4.118 – Implementação do switch

```
switch (bt){
    case 1: ajuda.setImageResource(R.drawable.imagem1); break;

    case 2: ajuda.setImageResource(R.drawable.imagem2); break;

    case 3: ajuda.setImageResource(R.drawable.imagem3); break;

    case 4: ajuda.setImageResource(R.drawable.imagem4); break;

    case 5: ajuda.setImageResource(R.drawable.imagem5); break;

    case 6: ajuda.setImageResource(R.drawable.imagem6); break;

    case 7: ajuda.setImageResource(R.drawable.imagem7); break;

    case 8: ajuda.setImageResource(R.drawable.imagem8); break;

    case 9: ajuda.setImageResource(R.drawable.imagem9); break;

    case 10: ajuda.setImageResource(R.drawable.imagem10); break;
```

No método *onClick* referente ao botão *Voltar* e no método *OnBackPressed*, como pode ser visto na Figura 4.119, será chamada a função *condicao*, tal função será explicada posteriormente, e o método *onClick* referente ao botão *Sair da Ajuda* simplesmente finalizará a tela presente.

Figura 4.119 – Método onClick e OnBackPressed

```
    btvoltar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            condicao();
        }
    });

    btsairaj.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            finish();
        }
    });

}

@Override
public void onBackPressed(){
    condicao();
}
```

A função *condicao* chamada no método *onClick* referente ao botão *Voltar* e no método *OnBackPressed* terá estruturas condicionais (*if*) com o valor de *bt*, pois como a tela com o recurso da imagem com ajuda é única para qualquer opção escolhida anteriormente, o valor do inteiro *bt*, que representa a opção escolhida, determinará a qual tela deve se retornar, como pode ser visto na Figura 4.120

Figura 4.120 – Determinando a qual tela deve se retornar

```
public void condicao (){
    if (bt <= 3){
        Intent intent = new Intent(AjudaImagem.this, Ajuda1.class);
        startActivity(intent);
        finish();
    }

    if (bt == 4 || bt == 5){
        Intent intent = new Intent(AjudaImagem.this, Ajuda2.class);
        startActivity(intent);
        finish();
    }

    if (bt > 5 && bt <= 8 ){
        Intent intent = new Intent(AjudaImagem.this, Ajuda3_1.class);
        startActivity(intent);
        finish();
    }

    if (bt > 8 ){
        Intent intent = new Intent(AjudaImagem.this, Ajuda3_2.class);
        startActivity(intent);
        finish();
    }
}
```

4.5.14 Tela Resultado

A tela *Resultado* será apresentada ao usuário quando o *quiz* for completado, tanto no *modo aleatório* quando no *modo campanha*. A tela apresentará um texto escrito “Parabéns”, uma mensagem que o modo escolhido para realizar o *quiz* foi concluído, uma mensagem informando a pontuação adquirida e um botão para finalizar a tela, como é ilustrado pela Figura 4.121.

Figura 4.121 – Tela Resultado

4.5.14.1 Arquivo activity_resultado.XML

O arquivo XML responsável pela parte visual da tela *Resultado*, possui O *LinerarLayout* como o gerenciador de *layout*, com os parâmetros *layout_weidth, layout_height, padding* e *orientation* sendo utilizados, como pode ser visto na Figura 4.122.

Figura 4.122 – Arquivo activity_resultado

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="10dp"
    android:orientation="vertical">
```

Um dos componentes do *layout* principal é um *layout* com três *TextView* destinado as mensagens “Parabéns” e “Seu aproveitamento foi de:”, como pode ser visto na Figura 4.123, e a mensagem que irá informar que o modo de jogo escolhido foi concluído.

Figura 4.123 – Componentes do arquivo activity_resultado

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1.5"
    android:gravity="center">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Parabéns!"
        android:layout_weight="1"
        android:id="@+id/titulo"
        android:layout_marginTop="20dp"/>

    <TextView
        android:id="@+id/subtitulo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_weight="1" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:layout_weight="8"
        android:text="Seu aproveitamento foi de:"
        android:id="@+id/textView"
        android:gravity="bottom" />

</LinearLayout>
```

O *layout* principal ainda tem outro componente *layout*, nele terá dois *TextViews's*, um destinado ao caractere *%* e outro destinado a pontuação conquistada realizando o *quiz*, como pode ser visto na Figura 4.124.

Figura 4.124 – Outros componentes do arquivo activity_resultado

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="2"
    android:gravity="center_horizontal">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:id="@+id/textResultado" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text=" %"
        android:id="@+id/textView10" />

</LinearLayout>
```

Outro componente do *layout* principal é um botão com o texto *OK*, como pode ser visto na Figura 4.125, destinado a encerrar a tela.

Figura 4.125 – Botão OK

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="OK"
    android:id="@+id/bt_ok" />
```

4.5.14.2 Classe Resultado.java

Na classe referente a tela *Resultado*, inicialmente são criados os objetos *CRUD* para manipulação do banco de dados, *resultado*, *conclusão* e *Bt_OK* para manipulação dos componentes do *layout* a ser utilizado como a parte visual e o objeto *cursor* para trabalhar com as informações do banco de dados referente ao usuário em questão, como pode ser visto na Figura 4.126. Para o método *setContentView* a referência do arquivo *activity_resultado* é passada. Os objetos criados são vinculados ao componentes aos quais se referem e *CRUD* é inicializado como objeto da classe *ManipulaBanco*.

Figura 4.126 – Criando os objetos e avariáveis da classe Resultado

```
ManipulaBanco CRUD;
TextView resultado, conclusao;
Button Bt_OK;
Cursor cursor;
int codigo, percTotal, modo;

@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_resultado);

    Bt_OK = (Button) findViewById(R.id.bt_ok);
    resultado = (TextView)findViewById(R.id.textResultado);
    conclusao = (TextView) findViewById(R.id.subtitulo);

    CRUD = new ManipulaBanco(getBaseContext());
```

Os inteiros *codigo* e *modo* receberão, respectivamente, os valores referentes à identificação do usuário e o modo de jogo escolhido para realizar o *quiz*, tais os valores

foram passados como parâmetros na troca de telas. O objeto *cursor*, como pode ser visto na Figura 4.127, receberá o retorno da função *carregarDadoById* passado o inteiro *codigo* como parâmetro.

Figura 4.127 – Cursor recebendo o retorno de carregarDadoById

```
codigo = this.getIntent().getIntExtra("codigo", 0);
modo = this.getIntent().getIntExtra("modo", 0);

cursor = CRUD.carregarDadoById(codigo);
```

Logo após, foram implementadas destruturas condicionais (*if*), como pode ser visto na Figura 4.128. Uma é destina caso o inteiro *modo* for igual a 1, ou seja, o modo campanha tenha sido escolhido para realizar o *quiz*, caso seja satisfeita essa estrutura condicional (*if*), a mensagem “Você concluiu o Modo Campanha do Show da Trigonometria!” será apresentada através do componente de *layout TextView* destinada a o campo de texto relacionado a pontuação conquistada.

Figura 4.128 – Implementação das estruturas condicionais

```
if(modo == 1) {

    conclusao.setText("Você concluiu o Modo Campanha \n" +
            "      do Show Da Trigonometria!");
    percTotal = cursor.getInt(cursor.getColumnIndexOrThrow(CriarBanco.TOTAL));

    resultado.setText(Integer.toString(percTotal));

}

if(modo == 2){

    conclusao.setText("Você concluiu o Modo Aleatório \n" +
                    "     do Show Da Trigonometria!");
    percTotal = cursor.getInt(cursor.getColumnIndexOrThrow(CriarBanco.TOTALA));

    resultado.setText(Integer.toString(percTotal));
}
```

Outra estrutura condicional (*if*) é caso o valor de *modo* for igual a 2, ou seja, o modo de jogo aleatório tenha sido escolhido. Neste caso a mensagem “Você concluiu o Modo Aleatório do Show da Trigonometria!” será apresentada através do componente de *layout TextView* a qual foi estipulado e o campo de texto destinado a pontuação conquistada também recebe a percentagem de perguntas acertadas

adquirida ao realizar o *quiz*.

Para o método *onClick* destinado ao botão *OK* e para o acionamento da tecla *retornar* do dispositivo, foi implementado a chamada da classe referente a tela *Perfil*, o envio da variável *codigo* passada como parâmetro na troca de telas e a finalização da tela *Resultado*, como pode ser visto na Figura 4.129.

Figura 4.129 – Implementação do método onBackPressed

```
    Bt_OK.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(Resultado.this, Perfil.class);
            intent.putExtra("codigo", codigo);
            startActivity(intent);
            finish();
        }
    });
}

@Override
public void onBackPressed() {
    Intent intent = new Intent(Resultado.this, Perfil.class);
    intent.putExtra("codigo", codigo);
    startActivity(intent);
    finish();
}
```

5 CONCLUSÃO E TRABALHOS FUTUROS

A motivação para utilizar o sistema operacional Android, no desenvolvimento do aplicativo *Show da Trigonometria*, se deu o fato de ele ser o mais utilizado pelos usuários de dispositivos móveis no mundo, possuindo o melhor posicionamento no mercado mundial. Além disso, existe um grande acervo de trabalhos feitos e tutorias sobre este sistema operacional.

Ao final do trabalho tem-se como resultado o aplicativo *Show da Trigonometria*, desenvolvido para o sistema operacional Android, utilizando as linguagens Java e XML com foco ao sistema superior a versão 4.4, conhecida como KitKat. O aplicativo permite o auxílio ao aprendizado do tema Trigonometria, na área da Matemática. Com o desempenho das realizações do jogo estando disponíveis, percebe-se a possibilidade de se realizar várias análises através dessas informações, sendo possível direcionar os estudos para determinado assunto com mais dificuldade.

Como trabalhos futuros tentem-se o teste do aplicativo aos alunos da disciplina Fundamentos de Matemática da UFVJM, solicitando a permissão do Conselho Nacional de Ética para realizar tal feito, sendo que não foi possível para esse trabalho pelo extenso tempo esperado para se obter a autorização. Também propõem-se uma nova versão do aplicativo com incremento dos outros temas, além da Trigonometria, lecionados na disciplina.

O professor atual da disciplina de Fundamentos de Matemática da UFVJM sugeriu como trabalhos futuros, estudar mais medidas de desempenho para se apresentar na aplicação e oferecer além das ajudas em imagens, também, ajuda em vídeo-aulas. O professor também sugere, que na próxima versão do jogo, o professor da disciplina tenha um modulo do aplicativo individual ao do aluno e estes serem utilizados de forma online, onde o aluno poderia tirar dúvidas mais especificas com o professor e este poderia acompanhar o desempenho dos seus alunos.

REFERÊNCIAS

ANDROID, 2016; Site Oficial. Disponível em <http://developer.android.com/>. Último acesso em: 3 fev. 2016.

AGUIAR, Denise Almeida. **O Ensino da matemática através de jogos nas séries inicias.** Disponível em: <http://pedagogiaaopedaletra.com/monografia-ensino-matematica-atraves-jogos-series-iniciais/>. Acesso em: 26 abr. 2015.

ALCANTRA, Carlos Augusto Almeida; VIEIRA, Anderson Luiz Nogueira. **Tecnologia móvel: uma tendência, uma realidade.** 2011. Disponível em: <http://arxiv.org/ftp/arxiv/papers/1105/1105.3715.pdf>. Acesso em: 17 mai. 2015.

BORIN, Júlia. **Jogos e resolução de problemas: uma estratégia para as aulas de matemática**. São Paulo: IME-USP, 2004

CLICK JOGO, 2015. Disponível em: <http://www.clickjogos.com.br/Jogos-online/Puzzle/Brain-Spa-Visual-Memory/>. Acesso em Abril de 2015

CHAVES, Aline Martins; SILVA, Gabriel. **Proposta de uma arquitetura de software e funcionalidades para implementação de um ambiente integrado de desenvolvimento para a linguagem php.** Bambuí, 2008. Disponível em: <http://www.cefetbambui.edu.br/str/artigos_aprovados/informatica/68-CO-5.pdf>. Acesso em: 03 fev. 2016.

CAMPOS, Sandra Gonçalves Vilas Boas; NOVAIS, Eliane Santana. **Jogos e brincadeiras para ensinar e aprender probabilidade e estatística nas séries iniciais do ensino fundamental.** In: Encontro Nacional de Educação Matemática, 10., 2010. Salvador. Anais..., Salvador: Sociedade Brasileira de Educação Matemática, 2010. Disponível em: <http://www.lematec.net/CDS/ENEM10/artigos/MC/T2_MC1938.pdf>. Acesso em: 17 mai. 2015.

DALLABONA, S. R; MENDES, S. M. S. **O lúdico na educação infantil: jogar, brincar, uma forma de educar.** Instituto Catarinense de Pós-Graduação, Disponível em: <http://www.posuniasselvi.com.br/artigos/rev04-16.pdf>. Acesso em: 17 mai. 2015.

EBAH, 2016; Disponível em <http://www.ebah.com.br/>. Último acesso em: 11 fev. 2016.

ECALCULO, 2016; Disponível em <ecalculo.if.usp.br/>. Último acesso em: 11 fev. 2016.

FANTACHOLI, Fabiane das Neves. **O brincar na Educação Infantil: Jogos, brinquedos e brincadeiras** - Um olhar Psicopedagógico. 5ª ed., Minas Gerais: Revista Científica Aprender, 2011. Disponível em: http://revista.Fundacaoaprender.Org.br/índex.php?id=148#mini. Acesso em: 20 de jun. de 2015.

FLANAGAN, David. **Java:** O Guia Essencial. Tradução Edson Furmakiewicz. 5. ed. Porto Alegre: Bookman, 2006.

FREIRE, Gabriela Moricone. **Software educacional: exemplos de aplicações em trigonometria no ensino médio.** Universidade Federal de Santa Catarina, Florianópolis, 2000. Disponível em: <https://repositorio.ufsc.br/bitstream/handle/123456789/97048/Gabriela_Moriconi_Freire.PDF?sequence=1>. Acesso em: 5 jun. 2015.

GEOESCOLA, 2016; Disponível em <http://www.geoescola.org/>. Último acesso em: 11 fev. 2016.

GRUBEL, Joceline Mausolff; BEZ, Marta Roseclear. **Jogos Educativos**. Revista Renote, v. 4, n. 2, 2006, p. 1-7. Disponível em: <http://seer.ufrgs.br/renote/article/view/14270>. Acesso em: 16 de jun. 2015.

IEZZI, Gelson. **Fundamentos de Matemática Elementar (Trigonometria)**. 9 ed. São Paulo: Atual Editora LTDA, 1995.

JUNIOR, Peter Jnadl. **Introdução ao Java**. São Paulo: Berkeley Brasil, 2002.

LACHETA, Ricardo R. **Google Android:** aprenda a criar aplicações para dispositivos móveis com Android SDK. 2 ed. São Paulo: Novatec Editora, 2010.

LOPES, José Marcos. **Uma Proposta para o Estudo de Conceitos Básicos de Probabilidade.** In: Congresso Nacional de Matemática Aplicada e Computacional, 32., 2009. Cuiabá. Anais..., Cuiabá: Sociedade Brasileira de Matemática Aplicada e Computacional, 1234 p. Disponível em: <http://www.sbmac.org.br/eventos/cnmac/xxxii_cnmac/pdf/203.pdf>. Acesso em: 23 mai. 2015.

MACHADO, Felipe Ribeiro. **Entretenimento Digital – Aspectos Mercadológicos.** Universidade Federal do Pernambuco, Recife, 2006. Disponível em: <http://www.cin.ufpe.br/~fab/cursos/metodologia-graduacao/2006-2/monografias/felipe-ribeiro.doc>. Acesso em: 11 abr. 2015.

MENDES. Douglas Rocha. **Programação Java com Ênfase em Orientação a Objeto.** São Paulo: Novatec. 2009

MONTEIRO, Juliana Lima. **Jogo, interatividade e tecnologia: uma análise pedagógica.** Universidade Federal de São Carlos. São Paulo. 2007. Disponível em: <http://www.ufscar.br/~pedagogia/novo/files/tcc/237167.pdf>. Acesso em: 11 abr. 2015.

MOUSQUER, Tatiana; ROLIM, Carlos Oberdan. **A utilização de dispositivos móveis como ferramenta pedagógica colaborativa na educação infantil.** URI, Santo Angelo (RS), [s.d.]. Disponível em: <http://www.santoangelo.uri.br/stin/Stin/trabalhos/11.pdf>. Acesso em: 17 mai. 2015.

OFICIAL DA NET, 2016. Disponível em <https://www.oficinadanet.com.br/post/13939-a-historia-do-android/>. Último acesso em: 15 jan. 2016.

OJOGOS, 2015. Disponível em: <http://www.ojogos.com.br/jogo/the-eyeballing-game>. Acesso em Abril de 2015

OKAMOTO, Sueli Ribeiro de Souza. **O jogo popular como conteúdo de ensino nas aulas de educação física.** Londrina, 2011. Disponível em: <http://www.uel.br/cef/demh/especializacao/doc/monografias/Sueli_Ribeiro.pdf>. Acesso em: 11 abr. 2015.

ORACLE, 2016; Site Oficial. Disponível em: < http://www.oracle.com/>. Último acesso em: 15 jan. 2016.

ORSO, Darci. Brincando, **Brincando Se Aprende**. Novo Hamburgo: Feevale, 1999.

PINTO, Claudia Simões, **Aplicando Brainstorming com apoio de Ferramenta Computacional**. Rio de Janeiro: Universidade Federal do Estado do Rio de Janeiro, 2007. Disponível em: <http://www.uniriotec.br/~pimentel/disciplinas/siscolab20072/Claudia/SC20072ArtigoClaudia>. Acesso em: 10 fev. 2015

PORVIR. **Jogo do MIT ensina ciência, engenharia e tecnologia.** Disponível em: <http://porvir.org/porcriar/jogo-mit-ensina-ciencia-engenharia-tecnologia/20130916> Acesso em: 1 fev. 2016.

RACHACUCA, 2015. Site Oficial. Disponível em: <http://rachacuca.com.br/quiz/matematica/>. Acesso em Abril de 2015.

RANGEL, Ana Cristina Teixeira Prado. **As Atividades Psicomotoras e o Desenvolvimento da Criança Cega Congênita.** Rio de Janeiro, 2009. Disponível em: <http://www.avm.edu.br/docpdf/monografias_publicadas/C203310.pdf>. Acesso em 10 jun. 2015.

RIBEIRO, Anecy Ruvieri; RIBEIRO, Benedito Aparecido; JUNIOR, Cleber Mena Leão. Capacitação Continuada: **O jogo como recurso pedagógico importante no processo ensino.** Disponível em: <http://www.pucrs.br/famat/viali/tic_literatura/jogos/Ribeiro.pdf>. Acesso em: 17 mai. 2015.

RIVED, 2016; Disponível em <http://rived.mec.gov.br/>. Último acesso em: 11 fev. 2016.

RODRIGUES, Gerusa Camargo; VAZ, Francieli Aparecida; OLIVEIRA, Cristiano Peres. **Avaliação do desempenho do curso de nivelamento em matemática na Universidade Federal do Pampa**. In: Encontro Regional de Estudantes de Matemática da Região Sul, 10., Bagé, 2014. Bagé: UNIPAMPA, 2014. p. 683-687.

SABOIA, Juliana; VARGAS, Patrícia Leal de; VIVA, Marco Aurélio de Andrade. **O uso dos dispositivos móveis no processo de ensino e aprendizagem no meio**

virtual. Revista Cesuca Virtual: conhecimento sem fronteiras. v.1, n.1, 2013. Disponível em: <http://ojs.cesuca.edu.br/index.php/cesucavirtual/article/viewFile/424/209>. Acesso em: 17 mai. 2015.

SANTOS, Josiel Almeida; FRANÇA, Kleber Vieira; SANTOS, Lúcia S. B. **Dificuldades na Aprendizagem de Matemática.** São Paulo, 2007. Disponível em: <http://www.educadores.diaadia.pr.gov.br/arquivos/File/2010/artigos_teses/MATEMATICA/Monografia_Santos.pdf/>. Acesso em: 17 de Nov. de 2015

SAVI, Rafael; ULBRICHT, Vania Ribas. **Jogos Digitais Educacionais: Benefícios e desafios**. Revista Renote, v. 6, n. 2, 2008, p. 1-10. Disponível em: <http://www.seer.ufrgs.br/renote/article/viewFile/14405/8310>. Acesso em: 18 mai. 2015.

SCHWABER, Ken; SUTHERLAND, Jeff. **The Definitive Guide to Scrum:** the rules of the game. Scrum.org, p.18, 2013. Disponível em: <https://www.scrum.org/scrum-guide>. Acesso em 02 fev. 2016.

SEBESTA, Robert W. Conceitos de linguagem de programação. 4.ed. Rio de Janeiro: Alta Books, 2000.

SEMANA DA TECNOLOGIA DA INFORMAÇÃO, 9. 2006, Fortaleza. **Java básico e intermediário.** Fortaleza: Insoft, 2006.

SILVA, Maycon Prado Rocha; COSTA, Paula Dornhofer Paro; PRAMPERO, Paulo Sérgio; FIGUEIREDO, Vera Aparecida de. **Jogos Digitais: definições, classificações e avaliação.** Campinas: Fac. De Eng Elétrica e de Computação, UNICAMP, 2009. Disponível em: <www.dca.fee.unicamp.br/~martino/disciplinas/ia369/trabalhos/t1g1.pdf>. Acesso em: 11 abr. 2015.

SUPER INTERESSANTE, 2016; Site Oficial. Disponível em <http://super.abril.com.br/conheca-a-historia-do-android-o-sistema-operacional-mobile-da-google>. Último acesso em: 15 jan. 2016.

UFF – UNIVERSIDADE FEDERAL FLUMINENSE, 2015; Site Oficial. Disponível em: <http://www.uff.br/>. Acesso em Abril de 2015.

UOL, 2015; Site Oficial. Disponível em: <http://jogos360.uol.com.br/math_mountain.html/>. Acesso em Abril de 2015

WADSWORTH, Barry. Jean Piaget para o professor da pré-escola e 1º grau. São Paulo, Pioneira, 1984.

ZANELLA, Matheus Koehler. **Representação Gráfica de Documentos XML.** Universidade Federal de Santa Maria, Santa Maria, 2011. Disponível em: <www-app.inf.ufsm.br/bdtg/arquivo.php?id=160&download=1>. Acesso em: 1 jan. 2016.

ANEXOS

ANEXO A – Entrevista com o professor da disciplina Fundamentos de Matemática lecionada na UFVJM

Entrevista realizada no dia 4 de Fevereiro de 2015, utilizando meio de comunicação virtual, tendo como participante o professor Alex Erickson Ferreira, atuando na Universidade Federal do Vales Jequitinhonha e Mucuri, sendo o atual professor da disciplina de Fundamentos de Matemática e tendo, também, o cargo de Vice-chefe do Departamento de Matemática e Estatística, com intuito de saber a atual situação do desempenho dos alunos na disciplina em questão e sua opinião e sugestão sobre o aplicativo desenvolvido neste trabalho. Vale ressaltar que o professor era livre para argumentar e desenvolver as respostas, demonstrando sua opinião e satisfação.

PERGUNTA 1: Para quantos e quais cursos é lecionado a disciplina de Fundamentos de Matemática?
RESPOSTA 1: "*Atualmente o curso de Fundamentos de Matemática é oferecido para os cursos de Sistemas de Informação, Química (licen.) e Biologia (licen.).*"

PERGUNTA 2: Quantas vezes/semestres você lecionou a disciplina Fundamentos de Matemática?
RESPOSTA 2: "*É um procedimento de praxe do DME atribuir a responsabilidade de uma disciplina – de todas as turmas – a um só professor. Salvo algumas exceções ou quando alguns professores efetivam trocas de disciplinas entre si.*
Sendo assim, todas as turmas estão sob minha responsabilidade desde o 1º. semestre de 2015, ou seja, há 3 semestres consecutivos.

Acho importante salientar que minha solicitação para lecionar a disciplina de Fundamentos de Matemática foi uma iniciativa pessoal de verificar o nível de conhecimento de matemática que os estudantes possuem ao ingressar na universidade.

Geralmente, as disciplinas que eu sempre lidei desde que cheguei na UFVJM foram Cálculo I e II (este último para o curso de Química) e GAAL. E foi devido ao

grande índice de reprovação e despreparo dos estudantes nestas disciplinas, que eu resolvi verificar pessoalmente o que estava ocorrendo.

É importante ressaltar que a disciplina de Fundamentos de Matemática é oferecida no primeiro período de cada curso."

PERGUNTA 3: Nesse período que você lecionou a disciplina, qual foi o índice de reprovação?

RESPOSTA 3: "*Este índice será interpretado como a quantidade de alunos reprovados em relação ao total de alunos da turma (matriculados regularmente), no entanto há algumas ressalvas a serem feitas posteriormente."*

Análise por turma:

Curso	*Período*	*Funções polinomiais*	*Exponencial e Logaritmo*	*Trigonometria*
Ciências Biológicas	Sem. 2014/2	6,85	5,33	3,68
	Sem. 2015/1	5	7,1	6,3
Química	Sem. 2014/2	6	4	2,4
	Sem. 2015/1	3,3	3,2	2,05
Sist. Informação	Sem. 2014/2	5,42	6	4,39
	Sem. 2015/1	5,7	6,7	7,3

Nesses índices não foi levado em consideração o número de desistentes e nem foi destacado os que obtiveram a aprovação passando pelo Exame Especial. Na verdade, poucos conseguem aprovação direta.

A questão da desistência se difere para cada curso na maioria dos casos. Por exemplo, o motivo mais casual que leva os alunos da Biologia a abandonarem a disciplina no meio do semestre letivo é a quantidade de obrigações que eles tem com as disciplinas específicas do próprio curso. A disciplina de Fundamentos não possui pré-requisito, sendo assim, eles podem fazê-la a qualquer momento. Logo, um número expressivo de alunos deixa para cursar essa disciplina no último semestre ou quando está prestes a formar.

Já os alunos de Química desistem mesmo do curso. E o motivo é o nível de dificuldade que eles encontram, também, em Fundamentos de Matemática."

PERGUNTA 4: Qual tema/prova tem a menor média de nota? (Pode-se informar as

medias de todas temas/provas se possível)

RESPOSTA 4: "*Observe a tabela a seguir.*

Curso	*Período*	*Funções polinomiais*	*Exponencial e Logaritmo*	*Trigonometria*
Biologia	2sem. 2014	6,85	5,33	3,68
	1sem. 2015	5	7,1	6,3
Química	2sem. 2014	6	4	2,4
	1sem. 2015	3,3	3,2	2,05
Sist. Informação	2sem. 2014	5,42	6	4,39
	1sem. 2015	5,7	6,7	7,3

Os assuntos escolhidos fazem parte da divisão natural que é estabelecida no plano de curso. Consequentemente, cada avaliação semestral é direcionada para cada tema supra definido na tabela. Está em estudo uma nova metodologia de cálculo para este índice, portanto não se deve atribuir a estes atuais outra interpretação senão um "mero reflexo" do tema em que os alunos tem melhor (ou pior) afinidade.

Por isso, não deve atribuir a esse índice algo do tipo "média das notas obtidas", pois existem outros fatores a serem considerados para este fim. Fatores estes que não foram levados em consideração neste momento.

Mesmo assim, estes índices refletem o que está sendo definido a priori como "afinidade com o conteúdo".

E por que está sendo utilizado o termo "afinidade"?

Porque a disciplina de Fundamentos de Matemática tem como incumbência fazer uma recapitulação de assuntos que já foram vistos (ou pelo menos deveria!) pelos alunos na sua trajetória estudantil passada. Em suma, nada do que é apresentado na disciplina é novidade. Aliás, a disciplina de Fundamentos é uma tentativa de uniformizar ou preparar o estudante para assuntos do nível superior, tais como o Cálculo.

Mas o que é observado na prática do dia a dia é a dificuldade dos estudantes de lidar com o pensamento que cada uma dos conteúdos exige.

Apesar a imprecisão dos índices, eles refletem bem o que se pode observar dentro da sala de aula e na resolução dos exercícios avaliativos (neste caso as provas semestrais)."

PERGUNTA 5: Por quais motivos você considera que esse tema/prova tem a menor

média de nota?

RESPOSTA 5: "*A base estudantil mal feita, ou seja, o 1º e o 2º grau concluído de maneira duvidosa*"

PERGUNTA 6: Você considera que Trigonometria é um tema da disciplina bem aceito pelos alunos durante as aulas?

RESPOSTA 6: "*Durante a prática docente, é comum ouvir praticamente de todos os estudantes, argumentos na qual eles expressão desconhecer ou terem tido quase nada a respeito do assunto.*

Na verdade, uma das tarefas mais difíceis durante o ensino da Trigonometria é mostrar a praticidade do assunto e, em seguida, migrar para o aspecto funcional do tema.

A Trigonometria tem esta característica.

Pode-se utilizá-la para resolver problemas do dia a dia e pode, também, utilizá-la numa abordagem de função para modelar fenômenos que possuem um comportamento periódico/sistemático, como as ondas sonoras ou estudos da ótica clássica.

A dificuldade maior é convencer os alunos a mudar o ponto de vista, ainda que estejamos lidando com as mesmas "ferramentas"."

PERGUNTA 7: Você considera a média das notas doa alunos no prova relacionada a Trigonometria uma média boa?

RESPOSTA 7: "*Pelo contrário. São PÉSSIMAS!*

E, sendo mais incisivo na questão, o disciplina de Fundamentos não consegue resolver o problema da falta de conhecimento básico do assunto.

Existem muitos fatores que contribuem para esse fracasso. Alguns ainda estão sendo analisados, pois há a esperança de que alguma coisa pode estar ao alcance da universidade, no entanto, ainda não tenho uma visão plena dos problemas. Consequente, isso demanda mais estudos sobre a questão."

PERGUNTA 8: Qual importância pra você do aplicativo *Show da Trigonometria* para

os alunos e o ensino da disciplina?

RESPOSTA 8: "*A ideia do aplicativo é ÓTIMA! Desde o início ela foi vista – por minha parte – com muita simpatia e, também, como um aliado para melhorar o aprendizado.*

Não diria que o aplicativo será a solução para uma questão que tem uma dimensão gigantesca e, ao mesmo tempo, de carácter heteromorfo.

Não obstante, toda tentativa que visa atacar o problema de forma construtiva é válida."

PERGUNTA 9: Quais sugestões você daria para trabalhos futuros envolvendo o *Show da Trigonometria*?

RESPOSTA 9: "*O aplicativo "Show da Trigonometria" é o primeiro passo que está sendo dado de forma contundente para se aliar a batalha do fraco ensino que as escolas de base vem fornecendo aos estudantes brasileiros.*

É também o uso da filosofia da tecnologia para auxiliar neste problema.

Mas ainda existem muitas perguntas a serem respondidas sobre essa abordagem. Algumas delas são:

- *Quais são as melhores medidas para medir o desempenho nesta atividade?*
- *Como fazer um aplicativo motivador para despertar no estudante a vontade de utilizá-lo?*
- *Além da ajuda fornecida pelo aplicativo, é possível associá-lo a vídeo-aulas sobre temas?*
- *O aplicativo pode estabelecer um "link" direto com o usuário (estudante) e seu professor para dúvidas mais específicas?*
- *O professor pode se cadastrar no aplicativo e, a partir daí, acompanhar o desempenho dos seus alunos "on line"?*

E talvez, o mais duvidoso e trabalhoso...

- *Qual é a melhor forma de apresentar o assunto e a sequência das atividades?*

Essa última questão só saberemos com o tempo e o trabalho."