UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ - UTFPR
MESTRADO PROFISSIONAL EM MATEMÁTICA EM REDE NACIONAL
PROFMAT

RENE AUGUSTO HANDA

DESENVOLVIMENTO DE APLICATIVOS COMO UMA FERRAMENTA DE APRENDIZAGEM NA ÁREA DE MATEMÁTICA

CURITIBA

2017

RENE AUGUSTO HANDA

DESENVOLVIMENTO DE APLICATIVOS COMO UMA FERRAMENTA DE APRENDIZAGEM NA ÁREA DE MATEMÁTICA

Dissertação apresentada ao Mestrado Profissional em Matemática em Rede Nacional da Universidade Tecnológica Federal do Paraná em Curitiba - PROFMAT-UTCT como requisito parcial para obtenção do grau de Mestre.
Orientador: João Luis Gonçalves

CURITIBA

2017

Dados Internacionais de Catalogação na Publicação

Handa, Rene Augusto
H236d Desenvolvimento de aplicativos como uma ferramenta de
2017 aprendizagem na área de matemática / Rene Augusto Handa.
-- 2017.
78 f. : il. ; 30 cm

Disponível também via World Wide Web
Texto em português, com resumo em inglês
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Mestrado Profissional em Matemática em Rede Nacional, Curitiba, 2017
Bibliografia: f. 77-78

1. Aplicativos móveis. 2. Matemática aplicada. 3. Computação – Matemática. 4. Internet das coisas. 5. Matemática – Estudo e ensino. 6. Matemática – Dissertações. I. Gonçalves, João Luis, orient. II. Universidade Tecnológica Federal do Paraná. Programa de Mestrado Profissional em Matemática em Rede Nacional. III. Título.

CDD: Ed. 22 – 510

Biblioteca Central da UTFPR, Câmpus Curitiba
Bibliotecária : Anna T. R. Caruso CRB9/935

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Pesquisa e Pós-Graduação

TERMO DE APROVAÇÃO DE DISSERTAÇÃO Nº 43

A Dissertação de Mestrado intitulada “Desenvolvimento de aplicativos como uma ferramenta de aprendizagem na área de matemática”, defendida em sessão pública pelo(a) candidato(a) Rene Augusto Handa, no dia 18 de dezembro de 2017, foi julgada para a obtenção do título de Mestre, área de concentração Matemática, e aprovada em sua forma final, pelo Programa de Pós-Graduação em Matemática em Rede Nacional.

BANCA EXAMINADORA:

Prof(a). Dr(a). João Luis Gonçalves - Presidente - UTFPR
Prof(a). Dr(a). André Fabiano Steklain Lisboa - UTFPR
Prof(a). Dr(a). Priscila Cardoso Calegari - UFSC

A via original deste documento encontra-se arquivada na Secretaria do Programa, contendo a assinatura da Coordenação após a entrega da versão corrigida do trabalho.

Curitiba, 18 de dezembro de 2017.

Carimbo e Assinatura do(a) Coordenador(a) do Programa

AGRADECIMENTOS

À Deus pois sem Ele nada seria possível, à minha esposa Michele pelo companheirismo e principalmente pela paciência, ao meu filho Caio por alegrar nossos dias e à minha mãe Ana por ser meu exemplo de força.

Ao meu pai Antonio (*in memoriam*) e à minha irmã, Marina (*in memoriam*) que estão comigo em todos os momentos.

Ao professor João Luis Gonçalves pela amizade, pelos ensinamentos, pela atenção e orientações.

Aos membros da banca examinadora pelos comentários, sugestões e contribuições, que ajudaram a melhorar a qualidade e a redação final do manuscrito.

“Não se turbe o vosso coração; credes em Deus, crede também em mim.
Na casa de meu Pai há muitas moradas;
se não fosse assim, eu vo-lo teria dito. Vou preparar-vos lugar.
E quando eu for, e vos preparar lugar, virei outra vez,
e vos levarei para mim mesmo para que onde eu estiver estejais vós também.”
(Bíblia Sagrada, João 14:1-3)

RESUMO

HANDA, Rene Augusto. **Desenvolvimento de aplicativos como uma ferramenta de aprendizagem na área de matemática**. 78 f. Dissertação - Programa de Mestrado Profissional em Matemática em Rede Nacional - PROFMAT, Universidade Tecnológica Federal do Paraná. Curitiba, 2017

Neste trabalho apresentamos a construção detalhada dos aplicativos: *Que Dia Foi*, que utiliza conceitos matemáticos de divisibilidade e congruência para determinar o dia da semana em que ocorreu determinada data e *Cálculo de Áreas*, que utiliza o Teorema do Cadarço para determinar a área de uma região definida por coordenadas coletadas via GPS. O desenvolvimento desses aplicativos requer habilidades que são comuns ao raciocínio matemático, além dos conceitos matemáticos inerentes aos aplicativos. Desta forma visamos oferecer uma opção para iniciativas de aprendizagem baseada em projetos, que é o desenvolvimento de aplicativos para dispositivos móveis com sistema operacional Android, através da plataforma on line MIT App Inventor 2. A plataforma MIT App Inventor 2 permite a programação por blocos que se encaixam, ou seja de forma mais visual e com uma série de recursos disponíveis.

Palavras-chave: MIT App Inventor 2. Desenvolvimento de Aplicativos. Matemática e Programação.

ABSTRACT

HANDA, Rene Augusto. **Applications development as a learning tool in mathematics**. 78 pg. Dissertation - Programa de Mestrado Profissional em Matemática em Rede Nacional - PROFMAT, Universidade Tecnológica Federal do Paraná. Curitiba, 2017

In this work we present a detailed construction of the following mobile applications: *Que Dia Foi* which uses mathematical concepts of divisibility and congruence to determine the day of the week in which a date occurred and *Cálculo de Áreas*, which uses the Shoelace Theorem to determine the area of a region defined by coordinates collected by GPS. The development of these mobile applications require skills that are common to mathematical thinking, as well as the mathematics concepts inherent in applications. In this way we aim to offer an option to project-based learning initiatives, which is the development of applications for mobile devices with Andriod operating system, through the online platform MIT App Inventor 2. The MIT App Inventor 2 platform allows to program by fitting blocks, that is, more visual and with a number of features available.

Keywords: MIT App Inventor 2. Applications Development. Mathematics and Programming.

LISTA DE ILUSTRAÇÕES

Figura 1 – Tela de permissão. 18
Figura 2 – Caixa de diálogo. 19
Figura 3 – Tela inicial. 19
Figura 4 – Menu *Connect*. 20
Figura 5 – Simulando o aplicativo. 20
Figura 6 – Menu *Build*. 21
Figura 7 – Elementos da tela de *Designer*. 21
Figura 8 – Conjuntos de blocos da seção *Palette*. 22
Figura 9 – Layout. 23
Figura 10 – Blocos da interface do usuário. 24
Figura 11 – Posicionamento dos blocos. 24
Figura 12 – Components. 25
Figura 13 – *Rename*. 25
Figura 14 – Configuração acima e esquerda. 26
Figura 15 – Configuração abaixo e à direita. 26
Figura 16 – Altura e largura. 27
Figura 17 – Exemplo de configuração da altura e largura. 27
Figura 18 – Textos dos componentes. 28
Figura 19 – *Hint*. 28
Figura 20 – Botão habilitado. 29
Figura 21 – Botão desabilitado. 29
Figura 22 – Teclado completo. 29
Figura 23 – Teclado numérico. 29
Figura 24 – Propriedades do *Spinner*. 30
Figura 25 – Diferenças entre as configurações dos alinhamentos horizontal e vertical. . . 30
Figura 26 – Botão e *Label*. 31
Figura 27 – *Listview*. 31
Figura 28 – *Spinner*. 31
Figura 29 – Criando elementos através da tela *Blocks*. 31
Figura 30 – Função do *textbox*. 32
Figura 31 – Função do *Notifier*. 32
Figura 32 – Funções do *LocationSensor*. 32
Figura 33 – Verificação se um número é par. 33
Figura 34 – Verificação se o número é par ou ímpar. 33
Figura 35 – Exemplo de aplicação da função *for*. 34
Figura 36 – Exemplo de operação matemática. 36

Figura 37 – *Modulo of*. 36
Figura 38 – Exemplo dos blocos de texto. 37
Figura 39 – Exemplo utilizando blocos de listas. 38
Figura 40 – Corrigindo erro da variável local. 38
Figura 41 – Exemplo utilizando blocos de procedimento. 39
Figura 42 – Exemplo utilizando blocos de procedimento. 39
Figura 43 – Bloco *Get*. 39
Figura 44 – Bloco *Set*. 39
Figura 45 – Alterar nomes e valores. 40
Figura 46 – Alterar função. 40
Figura 47 – Aba de alteração de função. 40
Figura 48 – Inline External Inputs. 41
Figura 49 – Colapse Block. 41
Figura 50 – *Mutator* ou Modificador. 41
Figura 51 – Funções *Get* e *Set*. 41
Figura 52 – Viewer do aplicativo. 47
Figura 53 – Algoritmo chave do mês. 49
Figura 54 – Algoritmo da contagem de anos bissextos. 50
Figura 55 – Verificação se um ano é bissexto. 51
Figura 56 – Configurando *Spinner* para o número de dias. 52
Figura 57 – Configurando o número de dias de acordo com o mês. 53
Figura 58 – Algoritmo *Que dia foi*. 55
Figura 59 – Configuração do botão BT_Verifique. 56
Figura 60 – Exemplo de triangularização. 59
Figura 61 – Paralelogramo ABCD. 59
Figura 62 – Teorema do Cadarço. 62
Figura 63 – Tela de *Designer* do aplicativo. 66
Figura 64 – Conversão de latitude e longitude em radianos. 66
Figura 65 – Algoritmo para o cálculo de M. 67
Figura 66 – Zona UTM e meridiano central. 68
Figura 67 – Cálculo de $NTCA$. 68
Figura 68 – Criação das listas de latitude e longitude. 69
Figura 69 – Cálculo da coordenada x. 69
Figura 70 – Cálculo da coordenada y. 70
Figura 71 – Lista das coordenadas x e y. 70
Figura 72 – Botão salvar. 71
Figura 73 – Teorema do cadarço. 72
Figura 74 – Trapézio de vértices (1,1), (2,3), (3,3) e (4,1). 73
Figura 75 – Botão Limpar. 73

Figura 76 – Alteração de posição do sensor. 74

SUMÁRIO

	INTRODUÇÃO	**13**
1	**MIT APP INVENTOR 2**	**16**
1.1	HISTÓRIA DO MIT APP INVENTOR 2	16
1.2	INICIAÇÃO AO MIT APP INVENTOR 2	17
1.3	TELA INICIAL	19
1.4	BARRA DE MENUS	19
1.4.1	Menu Connect	20
1.4.2	Menu Build	21
1.5	INICIANDO UM NOVO PROJETO	21
1.6	TELA DE DESIGNER	21
1.6.1	Seção Palette	22
1.6.1.1	*User Interface* e *Sensors*	22
1.6.1.2	*Layout*	23
1.6.2	Seção Viewer	24
1.6.3	Seção Components	25
1.6.4	Seção Properties	26
1.7	TELA BLOCKS	30
1.7.1	Blocos da tela de Designer	31
1.7.2	Blocos de controle	32
1.7.3	Blocos lógicos	35
1.7.4	Blocos matemáticos	35
1.7.5	Blocos de textos	37
1.7.6	Blocos de listas	37
1.7.7	Blocos de variáveis	38
1.7.8	Blocos de procedimentos	38
1.7.9	Blocos comuns entre conjuntos de blocos	39
1.8	FERRAMENTAS ÚTEIS	40
2	**APLICATIVO QUE DIA FOI**	**42**
2.1	MOTIVAÇÃO, DISCUSSÃO DO PROBLEMA E OBJETIVOS	42
2.2	MATEMÁTICA DO APLICATIVO QUE DIA FOI	42
2.3	IMPLEMENTAÇÃO DO APLICATIVO QUE DIA FOI	45
2.3.1	Configuração da tela de Designer	46
2.3.2	Ano selecionado	48
2.3.3	Chave do mês	48

2.3.4 Bissextos 50
2.3.5 O ano selecionado é bissexto? 51
2.3.6 Número de dias 52
2.3.7 Configuração do dia da semana 53
2.3.8 Definindo a função do botão Verifique 54

3 APLICATIVO CÁLCULO DE ÁREAS 57
3.1 MOTIVAÇÃO, DISCUSSÃO DO PROBLEMAS E OBJETIVOS 57
3.2 MATEMÁTICA DO APLICATIVO CÁLCULO DE ÁREAS 58
3.3 SISTEMA UTM 62
3.4 IMPLEMENTAÇÃO DO APLICATIVO CÁLCULO DE ÁREAS 64
3.4.1 Configuração da tela de Designer 65
3.4.2 Longitude, latitude em radianos 66
3.4.3 Cálculo de M 67
3.4.4 Zona UTM e meridiano central 68
3.4.5 Cálculo de NTCA 68
3.4.6 Lista de latitude e longitude 68
3.4.7 Cálculo de x 69
3.4.8 Cálculo de y 69
3.4.9 Lista das coordenadas e procedimento 70
3.4.10 Botão salvar 71
3.4.11 Botão Calcular e o Teorema do Cadarço 71
3.4.12 Botão limpar e alterações de latitude e longitude 73

4 CONCLUSÕES 75

REFERÊNCIAS 77

INTRODUÇÃO

Celulares e computadores já estão completamente atrelados ao cotidiano da sociedade e a tendência é que essa relação aumente com o tempo, por exemplo, espera-se uma revolução tecnológica que tem sido chamada de "*internet* das coisas", em que dispositivos eletrodomésticos, veículos e outros objetos serão "inteligentes" e interconectados e realizarão mais tarefas e de forma mais eficiente, inclusive sem a supervisão direta de seres humanos. Neste contexto construir aplicativos e *softwares* em geral é uma atividade de grande importância.

Os aplicativos de celulares fazem parte da rotina de grande parte da população, seja para saber a previsão do tempo, ler uma reportagem, informações do trânsito ou entretenimento. Porém, ainda não é tão comum que as pessoas desenvolvam seus próprios aplicativos, seja por desconhecimento, comodidade ou a dificuldade em lidar com a lógica de programação, mas, em um futuro próximo, isso tende a mudar, é o que afirmou Mitchel Resnick durante o evento Transformar, realizado em São Paulo, no ano de 2014. Mitchel Resnick é um dos diretores do grupo *MIT Media Lab*, pertencente ao MIT *(Massachusetts Institute of Technology)* e defende que a programação deveria ser tão importante quanto ler ou escrever e que em um mundo repleto de tecnologia, quem não aprender a programar será programado.

Segundo dados divulgados pela IDC, *International Data Corporation*, 95,5% comercializados no Brasil entre julho e setembro de 2016 utilizam o sistema operacional *Android* (HIGA, 2016, Acesso em: 10 dez. 2016).

O MIT possui uma plataforma gratuita e *on-line* cujo a versão atual é o *MIT App Inventor 2*, que possibilita o desenvolvimento de aplicativos para *Android*. A programação é desenvolvida por blocos com encaixes semelhantes aos de um quebra-cabeças, o que torna muito intuitiva a possibilidade ou não de combinação dos blocos e simplifica a iniciação à programação.

O *MIT App Inventor 2* tem como essência desenvolver aplicativos para celulares, porém, ao desenvolver programas, há uma série de habilidades que são aprimoradas implicitamente. Em 2016, uma escola de programação da Austrália, chamada *CS One Academy* utilizou, durante o programa de férias de julho, o *MIT App Inventor 2* como ferramenta para que seus alunos desenvolvessem essas habilidades (KAREN, 2016, Acesso em: 05 fev. 2017). Dentre essas habilidades destacamos:

- Reconhecimento de padrões: capacidade de, durante a resolução de algum problema, reconhecer situações semelhantes já vivenciadas anteriormente afim de auxiliar na resolução dos problemas atuais.
- Decomposição: decompor um problema de maior complexidade em diversos problemas de resolução mais simples.

- Organização: selecionar, dentre diversas informações quais são úteis e quais são dispensáveis e definir um conjunto de regras que sirvam como instruções de como usar as informações.
- Gerenciar eventos: a grande maioria dos algoritmos do *MIT App Inventor 2* consiste basicamente em gerenciar a ordem em que cada ação deve ser tomada de acordo com certas condições. A máquina não é capaz de agir de forma independente nesse contexto, é o programador que deve informar à máquina qual ação deve ser tomada.
- Procedimentos e estratégia: um programa bem sucedido não é feito de improviso. É preciso ter um método e seguir um planejamento. Durante o processo de criação de um aplicativo, algumas questões devem ser analisadas.
  - Qual o propósito do programa?
  - Como a máquina deve reagir sob determinada circunstância específica?
  - Todos os casos possíveis são considerados?

Esses conceitos são pertinentes a resolução de problemas e portanto devem também ser desenvolvidos no âmbito das aulas de matemática.

O curso de Recursos Computacionais no Ensino da Matemática é parte da matriz curricular do mestrado do PROFMAT. Destacamos dois trechos do livro texto (GIRALDO; CAETANO; MATTOS, 2013), homônimo à disciplina:

> No que diz respeito à integração de recursos computacionais na sala de aula de Matemática, temos como meta uma incorporação efetiva à prática docente - sem que o computador se reduza a um mero adereço, alegórico para a abordagem, e que a aula no laboratório de informática adquira um caráter de curiosidade, desconectada da aula "de verdade", aquela com quadro negro e giz.(GIRALDO; CAETANO; MATTOS, 2013)

> ... a questão a considerar não deve ser como recursos computacionais podem ser anexados a abordagem previamente estabelecidas, e sim como sua integração à prática docente pode viabilizar a produção de novas abordagens, possibilitando reestruturações da ordem e das conexões entre os conteúdos, e criando novas formas de explorar e de aprender Matemática.(GIRALDO; CAETANO; MATTOS, 2013)

Tomando como base os conceitos da disciplina supracitada, a relativa facilidade de se desenvolver aplicativos na plataforma do MIT, as habilidades que são desenvolvidas implicitamente no desenvolvimento de programas e a quantidade de celulares com sistema operacional *Android* em circulação no Brasil, este trabalho tem como proposta descrever detalhadamente as propriedades e como criar aplicativos usando o *MIT App Inventor 2*. Permeiam essa proposta a indissociabilidade entre as habilidades envolvidas na programação e no raciocínio matemático, bem como a descoberta da matemática envolvida nos produtos com tecnologia, em particular nos

aplicativos de celulares. Essa proposta está alinhada a aprendizagem baseada em projetos, em que a execução de projetos leva ao aprendizado de conteúdos, o protagonismo do aprendizado está no estudante e o maior interesse e motivação são consequências de um processo mais leve e natural.

Em resumo, o objetivo do trabalho é mostrar aos estudantes um meio de utilizar recursos tecnológicos, em particular a plataforma do *MIT App Inventor 2*, apresentando para isso, um guia de utilização com as principais ferramentas e funcionalidades da plataforma para desenvolver aplicativos relacionados com questões dos seus cotidianos e que nesse processo, os alunos aprendam matemática e percebam que a matemática está presente ao nosso redor, além de desenvolver as habilidades comuns a programação e a matemática. Nosso caminho pra esses objetivos é criar dois aplicativos e detalhar a programação e matemática que foi necessária. Os aplicativos desenvolvidos são *Que Dia Foi* e *Cálculo de Áreas*.

Esta dissertação está estruturada em três capítulos. No primeiro capítulo apresentamos um breve histórico sobre a origem do *MIT App Inventor 2*. Neste capítulo mostramos também como acessar a plataforma pela primeira vez, como iniciar um novo projeto, descrevemos os principais menus e blocos utilizados na elaboração dos aplicativos propostos neste trabalho, apresentando alguns exemplos simples de construções de algoritmos utilizando a plataforma.

No segundo capítulo mostramos o porquê escolhemos desenvolver o aplicativo *Que Dia Foi*, bem como a matemática envolvida no desenvolvimento do aplicativo proposto, posteriormente apresentamos um exemplo de implementação do aplicativo *Que Dia Foi*, utilizando a plataforma *MIT App Inventor 2*.

No terceiro capítulo propomos a construção do aplicativo *Cálculo de Áreas*, justificando sua pertinência, bem como a matemática utilizada no desenvolvimento do aplicativo e detalhamos como o aplicativo *Cálculo de Áreas* pode ser desenvolvido utilizando a plataforma *MIT App Inventor 2*.

1 MIT APP INVENTOR 2

A plataforma *MIT App Inventor 2*, criada pelo MIT, fundamenta-se no objetivo de popularizar o acesso à programação. Utiliza-se da linguagem de programação por blocos para desenvolver aplicativos para dispositivos *Android*, como a plataforma é *on-line* não é necessário a instalação de programas para criar os aplicativos.

Neste capítulo apresentamos uma breve história de como surgiu o *MIT App Inventor 2* (HARDESTY, 2010, Acesso em: 05 fev. 2017), descrevemos seus principais blocos, ferramentas e menus utilizados no desenvolvimento dos aplicativos propostos nesse trabalho. São descritos também os procedimentos necessários para instalar e testar no aparelho celular os aplicativos criados através desta plataforma.

1.1 HISTÓRIA DO MIT APP INVENTOR 2

A história do *MIT App Inventor 2* iniciou nos anos 60 quando o então professor do MIT, Seymour Papert juntamente com um grupo de alunos do MIT desenvolveram pesquisas com o objetivo de tornar viável o uso de computadores por crianças como instrumento de aprendizagem. Surgiu então a linguagem de programação *Logo*, que inicialmente se tratava de programar um robô em forma de tartaruga, que com uma caneta anexada a ele, fazia desenhos em uma folha de papel. Entretanto, naquela época os computadores tinham um custo extremamente elevado e era impensável que uma criança os utiliza-se como forma de entretenimento, ainda que lúdico, devido a isso, somente nos anos 80 com o início da popularização dos computadores pessoais é que utilizar a programação nas escolas tornou-se viável.

Mitchel Resnick e Eric Klopfer, atualmente professores e diretores de um programa educacional do MIT, juntamente com uma equipe do MIT, desenvolveram uma extensão do *Logo*, chamada *StarLogo*. O *StarLogo* é um ambiente de simulação que permite a interação do usuário com milhares de robôs virtuais semelhantes aos robôs do *Logo* original. Após concluir o projeto do *StarLogo*, Resnick começou a desenvolver um sistema semelhante ao *Logo* para que crianças programassem robôs construídos a partir dos blocos de montar como os da marca Lego, movidos de forma eletro-mecânica. Essa pesquisa culminou nos *kits Lego's Mindstorms*. Baseando-se nesse resultado, um graduando dessa época, chamado Andrew Begel desenvolveu uma linguagem de programação gráfica para tornar a programação mais intuitiva. Essa linguagem de programação gráfica é o precursor do *MIT App Inventor 2*.

O novo sistema de programação de Begel foi muito bem aceito. Então, Resnick e Eric Klopfer continuaram a desenvolver e chegaram em dois projetos distintos. O projeto de Resnick foi o Scratch, um desenvolvedor de jogos e histórias interativas para ambiente *Web* e o projeto de Klopfer resultou no *StarLogo TNG*, mais voltado para a criação de jogos.

Uma aluna chamada Ricarose Roque desenvolveu em seu mestrado uma versão mais geral do *StarLogo TNG*. Baseado nessa versão foram desenvolvidos os blocos de programação utilizados em um programa lançado em julho de 2010 pelo *Google*, chamado *Google App Inventor*, que tinha como objetivo viabilizar que pessoas sem experiência com programação pudessem desenvolver aplicativos para celulares que funcionem com o sistema operacional *Android*. O projeto do *Google App Inventor* foi liderado por um professor contratado do *Google*, chamado Hal Abelson que fez parte do grupo de alunos que participou da pesquisa de Resnick nos primórdios do *Logo*, ajudando-o inclusive a testar o *Logo* nas primeiras escolas.

No ano de 2011 o *Google* anunciou o fim de suas atividades com o *Google App Inventor* e o Centro de Aprendizagem Móvel, no MIT foi escolhido para hospedar um servidor público do *App Inventor* com a condição de que o *App Inventor* fosse um programa de código aberto. Em março de 2012 o MIT lança o *MIT App Inventor 1* e, finalmente, em dezembro de 2013, em um evento anual sobre ciências da computação é lançado o *MIT App Inventor 2*, que é a versão mais atual.

1.2 INICIAÇÃO AO MIT APP INVENTOR 2

No *MIT App Inventor 2*, o programador desenvolve seu aplicativo e há três formas de testá-lo, uma das maneiras é conectar o celular ao computador via cabo USB, a segunda maneira é emular o aplicativo no próprio computador, porém, se o aplicativo depender de recursos do aparelho celular, como câmera e sensores, não será possível testar totalmente o aplicativo. A terceira maneira é através de uma rede *Wifi*, uma vez que o computador e o celular estejam conectados à mesma rede, é possível fazer com que o *MIT App Inventor 2* gere um *QR code* ou um código de 6 letras que permitem emular o aplicativo em tempo real. Este último modo é altamente recomendado pelo *MIT App Inventor 2* (SETTING..., 2012, Acesso em: 10 fev. 2017)

Finalizados os testes, é possível instalar o aplicativo no celular, de duas maneiras. A primeira maneira é gerar um *QR code* que direciona para um *link* onde é possível baixá-lo diretamente no celular, a segunda maneira é baixar o aplicativo no formato de arquivo *.apk* no computador e posteriormente, instalá-lo no celular. Vale ressaltar que para instalar o aplicativo no celular é necessário habilitar uma opção de segurança que permita a instalação de aplicativos oriundos de fontes que não sejam a *Play Store*.

Os requisitos necessários para utilizar o *MIT App Inventor 2* são os seguintes:

- **Computador e sistema operacional**: um dos seguintes sistemas operacionais: *Windows XP*, *Windows Vista*, *Windows 7*, *Ubuntu 8* ou superior, *Debian 5* ou superior, *Mac OS X 10.5* ou superior.

  Obs: nas versões *Ubuntu* e *Debian 5* só é possível emular o aplicativo via rede *Wifi* e o *Mac OS* deve, necessariamente, utilizar processador *Intel*.

- **Navegador**: *Mozilla Firefox 3.6* ou superior, *Apple Safari 5.0* ou superior, *Google Chrome 4.0* ou superior.

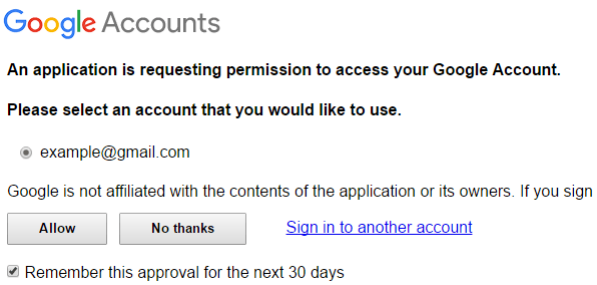
  Obs: Caso a extensão "*No Script*" do *Mozilla Firefox* estiver habilitada, é necessário desabilitá-la.

- **Celular ou *Tablet***: sistema operacional deve ser *Android 2.3(Gingerbread)* ou superior.

Sugerimos que o primeiro acesso à plataforma do *MIT App Inventor* seja realizado acompanhando simultaneamente à leitura dos procedimentos descritos nesta seção.

Para começar a programar, é preciso acessar a página *web ai2.appinventor.mit.edu*, que será redirecionada para realizar *logon* em uma conta do *Gmail*, caso o usuário não tenha, é necessário criá-la. Após realizar o *logon*, uma tela solicita a permissão para que o *MIT App Inventor 2* tenha acesso à conta, e é necessário permitir, clicando em *allow*, como apresentado na Figura 1. A cada 30 dias o *Mit App Inventor 2* solicitará a permissão para acessar a conta, desmarcando a opção *"Remember this approval for the next 30 days"* não será mais necessário autorizar o acesso.

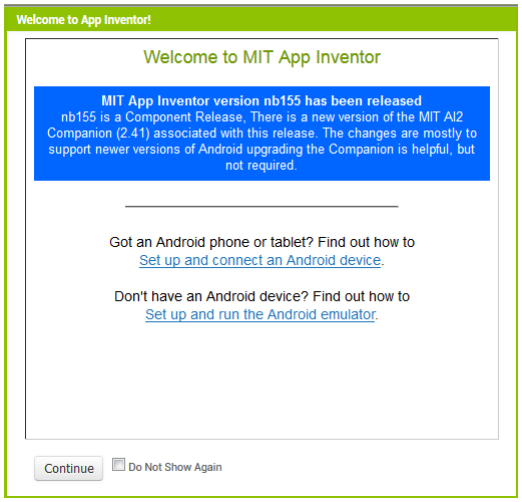
Figura 1 – Tela de permissão.



Após permitir o acesso à conta, é preciso aceitar os termos de serviço do *MIT App Inventor 2*. Na sequência uma caixa de diálogo é apresentada solicitando que o usuário responda uma pesquisa, é opcional respondê-la. Posteriormente é realizado o redirecionamento para a página do *MIT App Inventor 2*. Uma nova caixa de diálogo é mostrada, contendo *links* para pequenos tutoriais e algumas informações sobre as atualizações, caso não seja de interesse nenhuma dessa informações, basta clicar no botão *Continue*. A Figura 2 apresenta essa caixa de diálogo.

Uma terceira caixa de diálogo surgirá com uma breve explicação de como iniciar o primeiro projeto, para fechá-la basta clicar em algum lugar fora da caixa. Vale ressaltar que essas diversas caixas de diálogo aparecem por ser o primeiro acesso ao *MIT App Inventor 2*, os demais acessos tendem a ser mais diretos. Após fechar essa caixa de diálogo, é apresentada a tela inicial do *MIT App Inventor 2*.

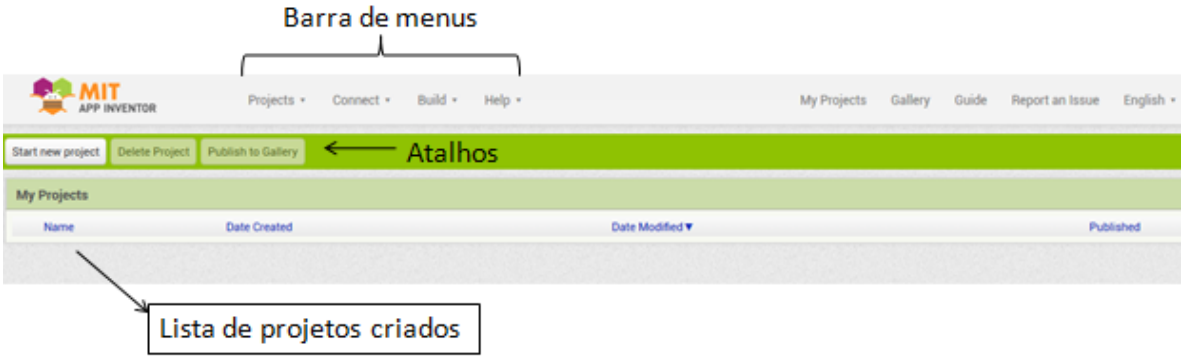
Figura 2 – Caixa de diálogo.



1.3 TELA INICIAL

A tela inicial é apresentada na Figura 3.

Figura 3 – Tela inicial.



Os objetos mais relevantes da tela inicial são a lista de projetos, os atalhos do menu e a barra de menus. Na lista de projetos constam todos os projetos salvos e os atalhos de menus são utilizados para iniciar um novo projeto, deletar um projeto ou publicar um projeto. Algumas funções importantes da barra de menus serão utilizadas na construção dos aplicativos. Essas funções serão descritas a seguir.

1.4 BARRA DE MENUS

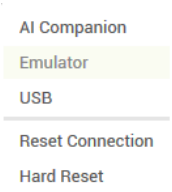
Para desenvolver os aplicativos deste trabalho, foram utilizados principalmente os menus *Projects*, *Connect* e *Build*. No menu *Project* é possível, dentre outras funções, iniciar, salvar, exportar, importar e deletar projetos. Nos menus *Connect* e *Build* é possível simular em tempo real e baixar os aplicativos, esses menus serão descritos a seguir.

1.4.1 MENU CONNECT

É através deste menu que é possível simular o aplicativo em tempo real. Como mencionado anteriormente, é possível realizar essa simulação de três modos diferentes. Como a própria página web do *MIT App Inventor 2* recomenda que seja simulado via rede*Wifi* é esse modo que será descrito a seguir. As outras formas de simular a execução do aplicativo são descritas em *http://appinventor.mit.edu/explore/ai2/setup.html*.

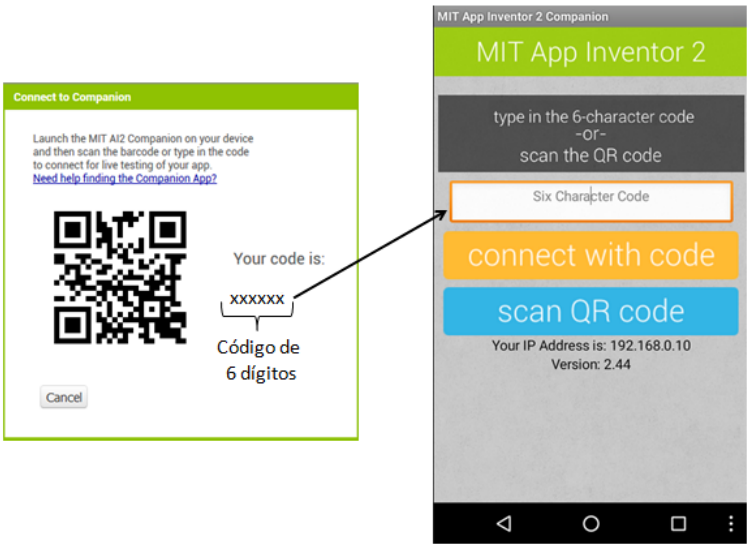
É necessário inicializar o aplicativo *MIT AI2 Companion* que pode ser baixado através da *Playstore*. Na plataforma do *MIT App Inventor 2*, clicar no menu *Connect* e, posteriormente, na opção *AI Companion*, como apresentado na Figura 4.

Figura 4 – Menu *Connect*.



Ao clicar nessa opção, será gerado um *QR code* e um código de 6 dígitos. Para iniciar a simulação, basta abrir o aplicativo *MIT AI2 Companion* no celular e, via *QR Code*, ou mesmo digitando o código de 6 dígitos, acessar o aplicativo, como apresentado na Figura 5.

Figura 5 – Simulando o aplicativo.



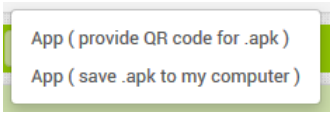
Após realizar esse procedimento a execução do aplicativo deve ser simulada no celular e as alterações que são realizadas devem ser atualizadas em tempo real. Para desconectar-se desse modo, basta clicar em *Reset Connection*.

Ao simular os aplicativos, constatamos a ocorrência de falhas de comunicação com o servidor do *MIT App Inventor 2*.Caso esse erro ocorra, o aplicativo é fechado no celular. Para reconectar bastar gerar um novo código seguindo os procedimentos descritos nesta subseção.

1.4.2 MENU BUILD

Clicando no menu *Build* as opções da Figura 6 são apresentadas.

Figura 6 – Menu *Build*.



Por meio deste menu é possível gerar um arquivo no formato .apk para instalar o aplicativo no celular. Esse arquivo pode ser gerado de duas formas, via *QR Code*, clicando na primeira opção ou baixando o arquivo no computador para posteriormente transferi-lo para o celular, clicando na segunda opção.

1.5 INICIANDO UM NOVO PROJETO

É possível iniciar um novo projeto por meio do menu *Project* ou clicando no atalho *Start new project*. Nesta etapa é necessário nomear o projeto, vamos nomeá-lo como Projeto1. Para a elaboração de um projeto são utilizadas duas telas, a tela de *Designer* e a tela *Blocks*, que no decorrer deste trabalho pode ser chamada de tela de blocos. Essas telas serão explicadas a seguir.

1.6 TELA DE DESIGNER

Ao iniciar um novo projeto, os elementos da tela de *Designer* pertinentes a esse trabalho são apresentados na Figura 7.

Figura 7 – Elementos da tela de *Designer*.

Estes elementos serão descritos a seguir:

- **Nome do projeto**: mostra o nome do projeto corrente, no nosso caso, Projeto1;
- **Adicionar e alternar entre *screens***: caso algum aplicativo necessite mais de uma *Screen* é possível adicionar, alternar e remover *screens* através dos botões indicados na Figura 7;
- **Alternar entre *Designer* e *Blocks***: através dos botões *Designer* e *Blocks* é possível alternar, respectivamente, entre entre as telas de *Designer* e *Blocks*;

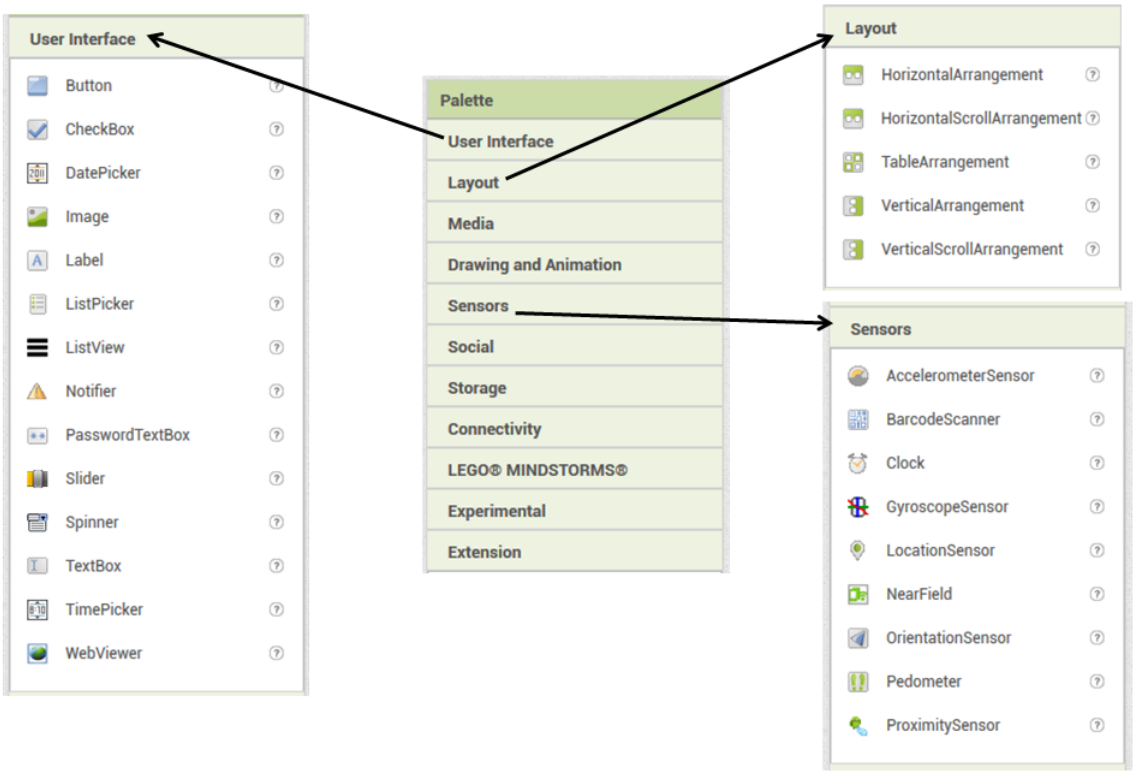
- **Seções *Palette*, *Viewer*, *Components* e *Properties***: são nessas seções que são configurados os elementos que serão utilizados nas construções dos aplicativos;

As seções *Palette*, *Viewer*, *Components* e *Properties* serão descritas a seguir.

1.6.1 SEÇÃO PALETTE

Local onde encontram-se os blocos utilizados para desenvolver a interface com o usuário. Os blocos são separados em conjuntos de blocos. Os conjuntos de maior utilidade para elaborar os aplicativos propostos nesse trabalho são: *User interface, Layout* e *Sensors*. Esses conjuntos de blocos estão indicados na Figura 8. A seção *User interface* pode ser eventualmente referida como interface do usuário posteriormente.

Figura 8 – Conjuntos de blocos da seção *Palette*.



Os conjuntos de blocos e respectivos blocos da seção *Palette* utilizados no desenvolvimento dos aplicativos deste trabalho serão descritos a seguir.

1.6.1.1 *USER INTERFACE* E *SENSORS*

Os blocos utilizados na construção dos aplicativos pertencentes ao conjunto *User interface* contidos neste trabalho são: *Button* que é um botão, *Label* que é uma legenda, *ListView* que é um visualizador de lista, *Notifier* que permite que seja apresentado um aviso na tela, como em caso de erros ou falhas por exemplo, *Spinner* que permite a criação de uma lista de elementos

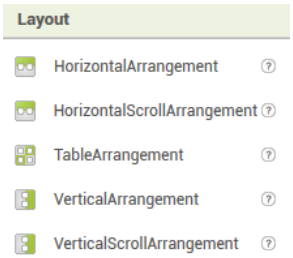
selecionáveis, e *TextBox* que é uma caixa de texto. Eventualmente podemos nos referir a *Button, Label, Notifier* e *TextBox*, respectivamente como botão, legenda, aviso e caixa de texto.

Do conjunto de blocos *Sensors*, apenas o bloco *LocationSensor* será utilizado na captura da latitude e longitude para o aplicativo Cálculo de Áreas.

1.6.1.2 *LAYOUT*

A função do conjunto é organizar os blocos de modo que a interface seja a mais clara e agradável possível. Ao clicar sobre o menu *Layout*, as opções da Figura 9 devem ser apresentadas.

Figura 9 – Layout.



Os blocos do *Layout* serão descritos a seguir.

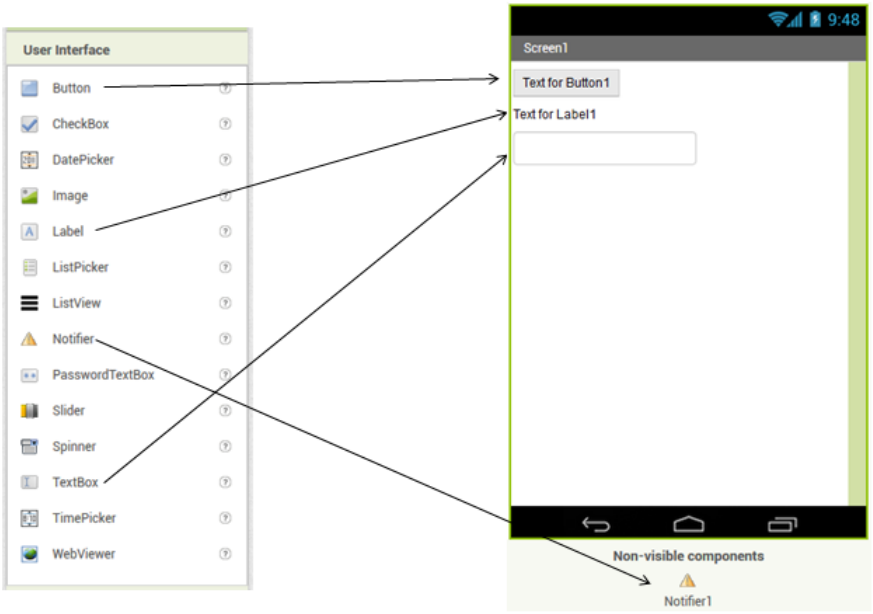
- ***HorizontalArrangement*** **e** ***HorizontalScrollArrangement*** : (arranjo horizontal) é possível dispor os blocos segundo uma linha horizontal é como se fosse uma linha com a possibilidade de inserir diversas colunas(blocos). A diferença entre esses dois blocos é que o *HorizontalScrollArrangement* habilita uma barra de rolagem quando o número de blocos inseridos na linha ultrapasse o limite da tela do celular ou do emulador, o que não ocorre no *HorizontalArrangement*.
- ***TableArrangement***: (arranjo em tabela) disponibiliza a organização dos blocos em uma tabela.
- ***VerticalArrangement*** **e** ***VerticalScrollArrangement*** : (arranjo vertical) semelhante ao arranjo horizontal dos blocos, porém neste, a organização é realizada na vertical.

Para exemplificar, serão dispostos alguns blocos de modo a desenvolver um aplicativo genérico onde o usuário informa um número através de uma caixa de texto, e após pressionar um botão, o aplicativo analisa esse número e retorna o resultado da análise em um *label*, será também utilizado um *Notifier* para avisar o usuário caso algum problema seja detectado.

Inicialmente, é preciso arrastar um botão, um *label*, uma caixa de texto e um *Notifier* para a seção *Viewer*, como indicado na Figura 10.

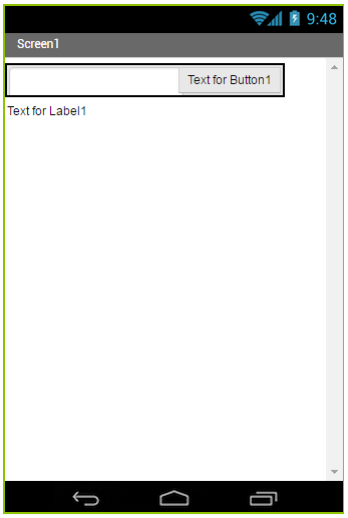
Será alterada a disposição dos blocos, pode-se por exemplo, arrastar o bloco de *Layout*, *HorizontalArrangement* para a seção *Viewer*. Deve-se então arrastar o bloco caixa de texto para

Figura 10 – Blocos da interface do usuário.



dentro do arranjo horizontal, feito isso, é preciso arrastar o *button* direcionando-o para o lado direito da caixa de texto, afim de que o os blocos fiquem posicionados como na Figura 11.

Figura 11 – Posicionamento dos blocos.



A continuação da construção do aplicativo Projeto1 será realizada nas seções posteriores.

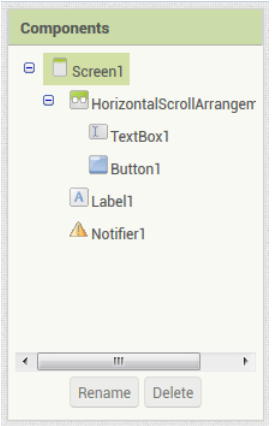
1.6.2 SEÇÃO VIEWER

A seção *Viewer* simula como ficará a tela do celular conforme os blocos vão sendo criados. Para adicionar novos blocos na construção dos aplicativos, basta arrastá-los à seção *Viewer*.

1.6.3 SEÇÃO COMPONENTS

Nesta seção é possível verificar a estrutura geral da disposição dos blocos e ainda, renomear e deletar blocos, a Figura 12 mostra a disposição do nosso aplicativo, Projeto1.

Figura 12 – Components.



Note que a *"Screen1"* é o componente mais à esquerda, à sua direita encontram-se os blocos *"HorizontalScrollArrangement1"*, e *"Label1"*. Esses blocos estão à direita porque estão contidos na *"Screen1"*, por esse mesmo motivo, os blocos *"TextBox1"* e *"Button1"* estão à direita do arranjo horizontal.

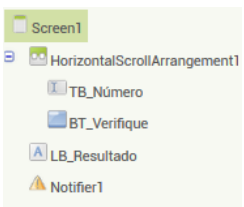
É possível minimizar a estrutura de blocos, ou parte dela, clicando no botão "-" e expandir clicando no botão "+". É possível observar ainda, na Figura 12 que há uma barra de rolagem, para o caso em que a estrutura de blocos ultrapasse o limite das bordas desta seção. Abaixo desta barra de rolagem, há os botões *"Rename"* e *"Delete"*, cujo as funções são renomear e deletar blocos, respectivamente. Sua utilização será descrita a seguir:

Botão *Rename*: o nome padrão dos blocos criados segue o raciocínio: se foram criados n blocos *TextBox*, o próximo bloco desse tipo será nomeado automaticamente para *"TextBox*$(n+1)$*"*. O botão *Rename* é utilizado para renomear blocos.

Em aplicativos que utilizem diversos blocos iguais, é interessante renomeá-los adequadamente, pois por exemplo, um aplicativo que use, dentre outros blocos, três *TextBox* um deles para que o usuário digite um nome, outro para que se digite um número e outro para que se digite um percentual, ocorrerá que os *TextBox* serão nomeados automaticamente como *"TextBox1"*, *"TextBox2"* e *"TextBox3"*. Para evitar possíveis equívocos entre as entradas, é possível renomear as *TextBox* como, por exemplo, "TB_nome", "TB_número" e "TB_percentual" onde optamos por utilizar TB para se referir à *TextBox* e, seguido do *underline*, um nome que indique a finalidade do bloco. A nomenclatura blocos dos aplicativos presentes nesse trabalho seguirá esse raciocínio, porém, esse padrão adotado é somente uma sugestão.

No aplicativo Projeto1, os blocos foram renomeados conforme a Figura 13.

Figura 13 – *Rename*.



Com essa nomenclatura, tem-se uma noção de que a caixa de texto, chamada de "TB_Número", será utilizada como entrada de um valor numérico, o botão "BT_Verifique" envia um comando para que o aplicativo verifique o número e a legenda "LB_Resultado", servirá para o aplicativo comunicar o resultado da sua verificação.

Obs.: Os nomes alterados na seção *Components* serão mostrados somente ao programador. Para definir o nome que o bloco apresentará para o usuário, utiliza-se a seção *Properties*, apresentada na próxima seção.

Botão *Delete*: Este botão tem como função excluir algum bloco. Ao clicar no botão *Delete*, uma caixa de diálogo será mostrada confirmando a ação, para consentir, basta clicar em *OK*. É importante ressaltar que ao excluir um bloco que tenha outros componentes em seu interior, é excluído o conjunto todo desses blocos, por exemplo, se no aplicativo Projeto1 o bloco *HorizontalArrangement* fosse apagado, os blocos"TB_Numero" e "BT_Verifique" também seriam.

1.6.4 SEÇÃO PROPERTIES

Clicando sobre um componente, é possível configurar suas propriedades. Devido às diferentes funções de cada bloco, podem haver propriedades diferentes também. A seguir serão descritas as propriedades que são comuns a alguns blocos e posteriormente, serão apresentadas as propriedades específicas do botão, da caixa de texto e dos *Spinners*.

***AlignHorizontal* ou alinhamento horizontal**: define o alinhamento horizontal de um ou mais objetos situados no interior do componente selecionado, sendo que *left* posiciona o objeto à esquerda e *right* posiciona o objeto à direita.

***AlignVertical* ou alinhamento vertical**: define o alinhamento vertical de um ou mais objetos situados no interior do componente selecionado, sendo que *top* posiciona o objeto acima e *bottom* posiciona o objeto abaixo.

Por exemplo, no Projeto1 que está sendo desenvolvido, ao alinhar horizontalmente o componente *Screen1* como *left* e alinhar verticalmente como *top*, os componentes posicionam-se acima e à esquerda, como na Figura 14 e ao alinhá-lo horizontalmente como *right* e verticalmente como *bottom*, os componente são posicionados abaixo e à direita, como na Figura 15.

Figura 14 – Configuração acima e esquerda.

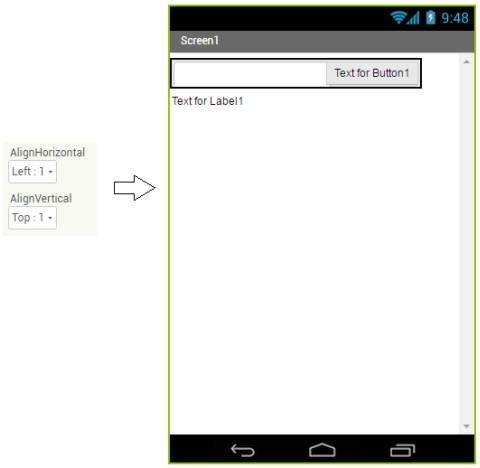
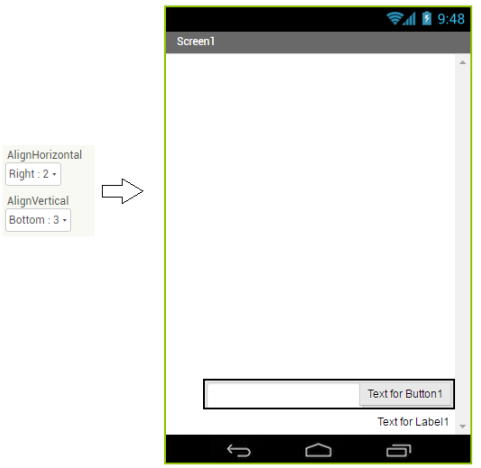
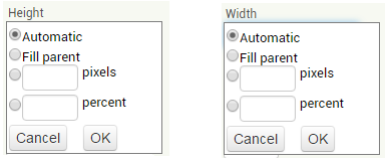


Figura 15 – Configuração abaixo e à direita.



Height* e *Width: As propriedades de *Height* e *Width* definem, respectivamente, a altura e a largura de um componente. As opções disponíveis são *Automatic*, *Fill parent* e configurar manualmente através de *pixels* ou percentual. As opções possíveis são apresentadas na Figura 16 e descritas a seguir.

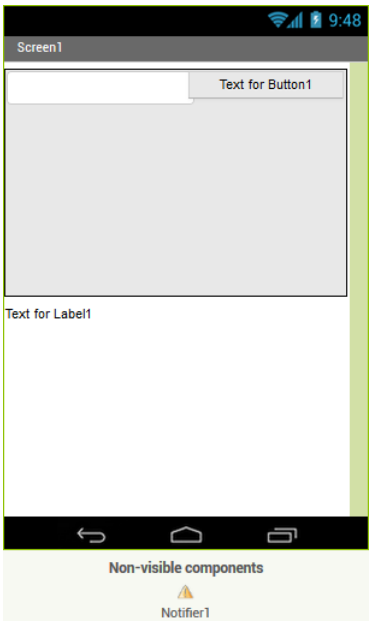
Figura 16 – Altura e largura.



- *Automatic*: ou automático, a altura ou largura são definidos automaticamente pelo sistema. Todos os componentes criados tem como padrão, a altura e largura definidos como automático.
- *Fill parent*: o componente em questão preenche todo o espaço disponível.
- Configuração manual: define-se, por *pixels* ou um valor percentual, o tamanho que o componente ocupará na tela.

Para exemplificar, no aplicativo Projeto1, selecionamos o componente *HorizontalArrangement1* e configuramos sua altura para 50% e largura para *Fill Parent*. Selecionamos o componente "BT_Verifique" e alteramos a largura para *Fill Parent*. Após realizar esses passos, a seção *Viewer* é apresentada na Figura 17.

Figura 17 – Exemplo de configuração da altura e largura.



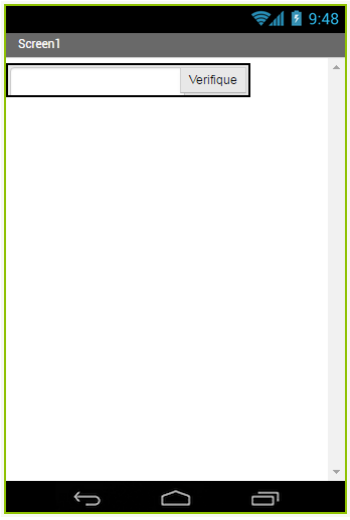
Como a alteração dessas configurações servem somente como exemplo, pode-se retornar os blocos alterados para a configuração *automatic*.

Text: diferentemente do *Rename* da seção *Components*, o *Text* altera o nome que o componente apresentará ao usuário. Na Figura 17, é possível observar que sobre o botão está

escrito *Text for Button 1* e que no *label* está escrito *Text for Label 1*. Caso não sejam alterados os nomes desses componentes, serão esses os textos que aparecerão para o usuário.

Como o botão será utilizado para realizar uma verificação, é adequado para o usuário, que o texto apresentado a ele seja "Verifique", por exemplo. Em nossa construção o botão mostrará o texto "Verifique" e o *label* não terá texto algum, pois servirá somente para mostrar o resultado.

Figura 18 – Textos dos componentes.

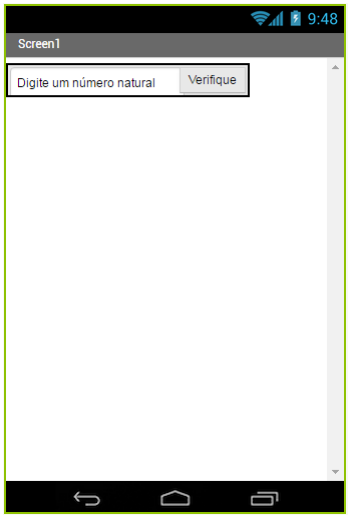


Para realizar essas alterações no Projeto1, basta selecionar o botão "BT_Verifique" e na seção *Properties*, clicar em *Text for Button 1*, na opção *Text*, alterar o nome para "Verifique". Para o *label*, o procedimento é análogo, porém, deve-se apenas apagar o *Text for Label 1*, deixando esse espaço em branco. Realizado essas alterações, a seção *Viewer* deve apresentar a tela como na Figura 18.

É possível observar que o *label* não aparece mais na tela, ele mostrará apenas o resultado do aplicativo, ou seja, mostrará somente o resultado da verificação.

Hint: serve como um auxílio, uma dica para o usuário sobre um determinado componente. Por exemplo, no aplicativo Projeto1, a caixa de texto "TB_Numero" foi criada para que o usuário digite um número natural qualquer, é plausível então, informar o usuário sobre a finalidade desse componente.

Figura 19 – *Hint*.



Para alterar essa propriedade, selecionamos o componente "TB_Numero" e, na propriedade *Hint*, escrevemos "Digite um número natural"e pressionamos *Enter*. A tela em *Viwer* ficará como na Figura 19.

Propriedade *Enable* do botão: para o aplicativo Cálculo de Áreas foi utilizada a propri-

edade *Enable* do botão, para alterá-la, basta marcar/desmarcar a opção *Enable*. Essa propriedade habilita ou desabilita o botão. As Figuras 20 e 21 a seguir, apresentam a alteração promovida pela propriedade *Enable*.

Figura 20 – Botão habilitado.

Figura 21 – Botão desabilitado.

Propriedade *NumbersOnly* da caixa de texto: Para o desenvolvimento do aplicativo *Que Dia Foi* utilizou-se a propriedade *NumbersOnly*. Para alterar essa propriedade, basta marcar a opção *NumbersOnly*. Alterando essa propriedade, apenas o teclado numérico é disponibilizado ao usuário. As Figuras 22 e 23 a seguir, apresentam a alteração promovida pela propriedade *NumbersOnly*.

Figura 22 – Teclado completo.

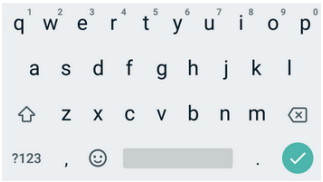
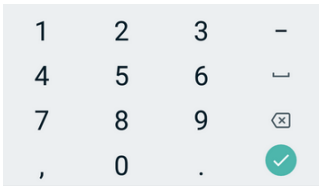


Figura 23 – Teclado numérico.



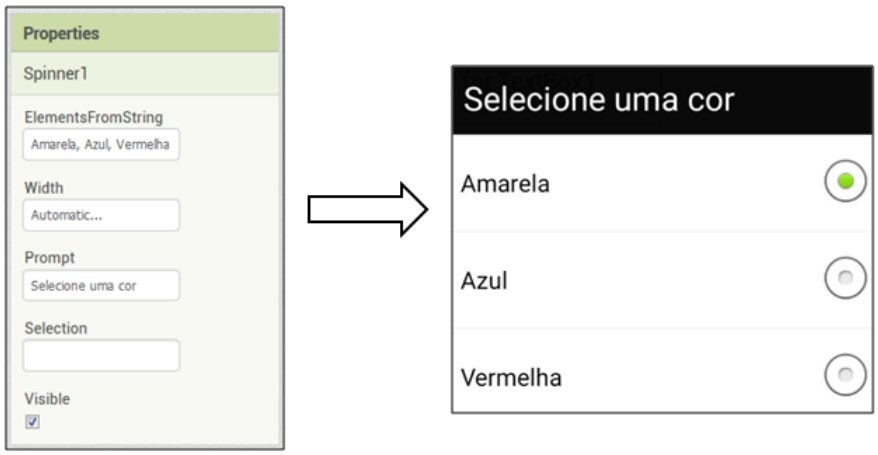
Propriedades do *Spinner*: Para o desenvolvimento do aplicativo *Que Dia Foi* foram configuradas as propriedades *ElementsFromString* e *Prompt*. Essas propriedades serão descritas a seguir.

- *ElementsFromString*: é possível criar uma lista com componentes selecionáveis através dessa propriedade. O primeiro elemento da lista deve ser um elemento não-selecionável pois a função *AfterSelecting* que será vista posteriormente, não reconhece o primeiro elemento como uma seleção válida. Os elementos da lista devem ser separados por vírgulas.
- *Prompt*: é possível definir o título do *Spinner* através dessa propriedade.

Para exemplificar, será criado um *Spinner* com três opções de cores: amarela, azul e vermelha e o título desse *Spinner* será: “Selecione uma cor”. Para configurar o *Spinner* dessa forma, basta digitar “Amarela, Azul, Vermelha” na propriedade *ElementsFromString* e “Selecione uma cor” na propriedade *Prompt*, então, quando o *Spinner* for mostrado na tela para o usuário, deverá ser da forma como apresenta a Figura 24.

Para que o *Spinner* seja apresentado desse modo para o usuário e para desenvolver a lógica dos aplicativos, é utilizada a tela *Blocks*, ou tela de blocos, que explicaremos a seguir.

Figura 24 – Propriedades do *Spinner*.



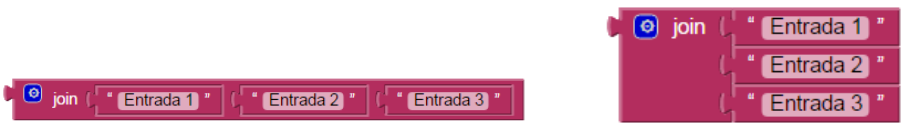
1.7 TELA BLOCKS

Para descrever e exemplificar os blocos do *MIT App Inventor 2*, o termo “Entrada” ou “Entrada(s)” será utilizado diversas vezes. Descreveremos o modo que nos referiremos às entradas a seguir:

Entradas: este termo refere-se a um dado fornecido, seja este dado fornecido pelo usuário, por um bloco ou conjunto de blocos. Comumente as entradas citadas serão numeradas. No decorrer deste trabalho, a contagem será sempre feita da esquerda para a direita, caso as entradas estejam dispostas na horizontal (*Inline inputs*) e de cima para baixo, caso as entradas estejam dispostas na vertical (*External inputs*). Os termos *Inline inputs* e *External inputs* são ferramentas do *MIT App Inventor 2* que serão descritas posteriormente.

Por exemplo, suponha que um bloco está recebendo três entradas, quando cita-se “entrada 1”, “entrada 2” e “entrada 3”, a ordem dessas entradas deve ser entendida como apresenta a Figura 25, na horizontal e vertical, respectivamente.

Figura 25 – Diferenças entre as configurações dos alinhamentos horizontal e vertical.



Além dos blocos da tela de *Designer*, os blocos do *MIT App Inventor 2* são separados em oito grandes conjuntos de blocos, são eles: blocos de controle, lógicos, matemáticos, textos, listas, cores, variáveis e procedimentos. Os blocos utilizados no desenvolvimento desse trabalho serão descritos a seguir.

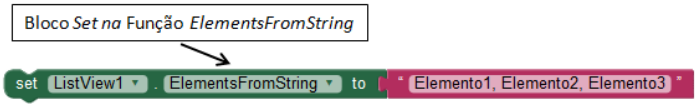
1.7.1 BLOCOS DA TELA DE DESIGNER

As ações dos blocos criados na tela de *Designer* podem ser configuradas na tela *Blocks*. As ações utilizadas na elaboração dos aplicativos deste trabalho serão descritas a seguir:

Figura 26 – Botão e *Label*.

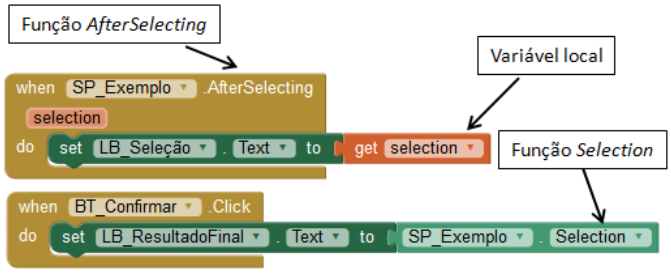
Funções do botão, *label* e caixa de texto: a Figura 26 apresenta um exemplo de aplicação da função *Click* e da função *Enable* do botão e da função *Text* do *label*: quando o botão "BT_Confirmar" for clicado, o algoritmo salva o texto do *label* "LB_Resultado" para 5, o botão "BT_Salvar" é habilitado e o botão "BT_Cancelar" é desabilitado. A função *text* também pode ser utilizada para o bloco caixa de texto.

Figura 27 – *Listview*.



Função do *listview*: a Figura 27, apresenta a propriedade análoga e homônima dos *Spinners*, porém, a lista de elementos é criada na tela *Blocks*.

Figura 28 – *Spinner*.



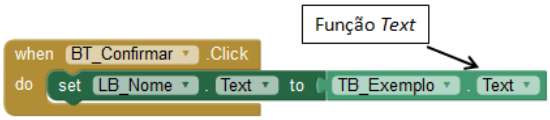
Funções do *Spinner*: a Figura 28 apresenta as funções *AfterSelecting* e *Selection* do *Spinner* que operam da seguinte forma: o *label* "LB_Seleção" recebe o valor da variável local *selection* que corresponde à seleção do *Spinner*. Quando o usuário clicar o botão "BT_Confirmar", o *label* "LB_ResultadoFinal" recebe o valor do elemento selecionado no *Spinner* "SP_Exemplo".

O *Spinner* também pode ter seus elementos criados na tela *Blocks*, como mostra a Figura 29.

Figura 29 – Criando elementos através da tela *Blocks*.

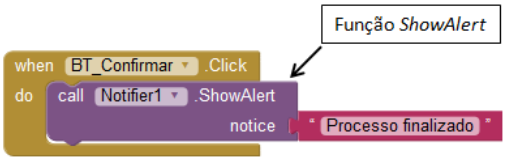
Vale ressaltar que no exemplo da Figura 29, o "Elemento1" correspondente a lista de elementos do *Spinner* "SP_Exemplo" deve ser não-selecionável.

Figura 30 – Função do *textbox*.



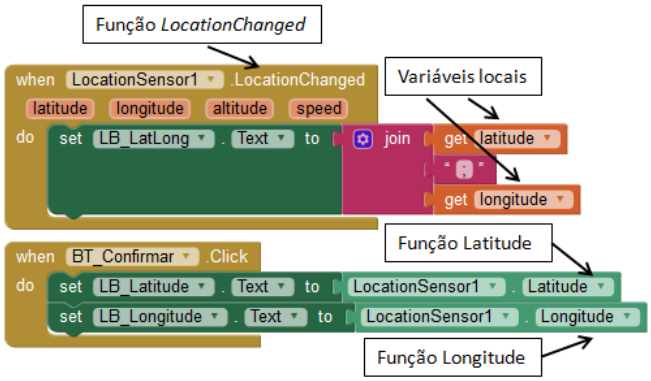
Função da caixa de texto: a Figura 30 apresenta a função *Text* da caixa de texto que salva o conteúdo presente na caixa de texto no *label* "LB_Nome" quando o botão "BT_Confirmar" for clicado.

Figura 31 – Função do *Notifier*.



Função do bloco *Notifier*: no algoritmo da Figura 31 a função *ShowAlert* envia o texto "Processo finalizado" para a tela do celular, quando o botão "BT_Confirmar" for clicado.

Figura 32 – Funções do *LocationSensor*.



Funções do *LocationSensor*: serão utilizadas três funções do *LocationSensor*, são elas: *LocationChanged*, *Latitude* e *Longitude*. Um exemplo do uso dessas funções pode ser observada na Figura 32. A função *LocationSensor* possui quatro variáveis locais implícitas, que alteram seus valores sempre que uma mudança na localização do GPS é reconhecida, desse modo, o *label* "LB_LatLong" recebe os valores correspondentes às variáveis latitude e longitude, separados por um ponto e vírgula.

As funções *Latitude* e *Longitude* do *LocationSensor* correspondem aos valores de latitude e longitude aferidos pelo GPS. A diferença das funções *Latitude* e *Longitude* para as variáveis locais implícitas no bloco *LocationSensor* é que as variáveis locais podem ser utilizadas apenas no interior do bloco *LocationSensor*. Na Figura 32, quando o botão "BT_Confirmar" é clicado os *labels* "LB_Latitude" e "LB_Longitude" são salvos com os respectivos valores de latitude e longitudes verificados pelo GPS.

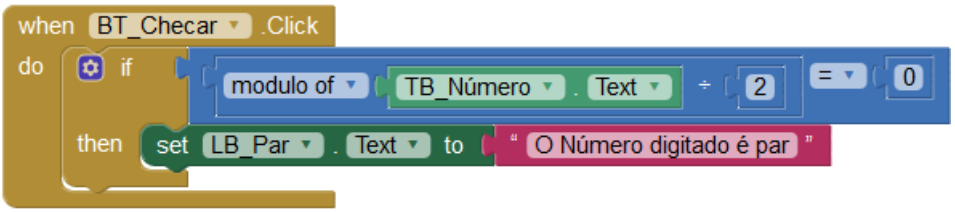
1.7.2 BLOCOS DE CONTROLE

Bloco *If*: o bloco conectado ao *"if"* é chamado de condição, se esta condição for verdadeira, então a ação definida no bloco conectado ao *"then"* é realizada, caso contrário, nada acontece. Este bloco possui a função modificadora, que permite acrescentar a função *"else"* e a função *if else*.

Exemplo: O algoritmo da Figura 33 verifica se um número é par.

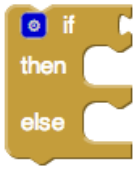
Este algoritmo funciona da seguinte forma: quando o botão BT_Checar for clicado, o

Figura 33 – Verificação se um número é par.



aplicativo deve verificar se o número contido em TB_Numero deixa resto igual a zero na divisão pelo número 2, caso isso seja verdade, o texto "O número digitado é par" é apresentado no *label* "LB_Par" se a divisão deixar resto diferente de zero, nada deve ser realizado.

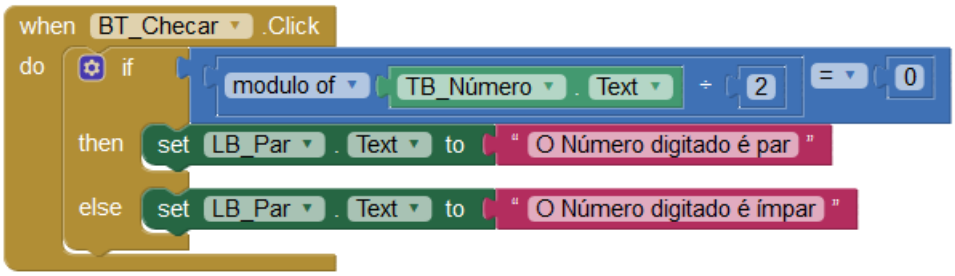
O bloco *modulo of* utiliza o conceito de congruência, que será apresentado posteriormente.



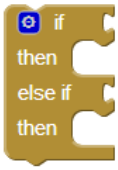
Bloco *IfElse*: semelhante ao bloco anterior, porém, quando a condição não é verdadeira, a ação definida no bloco conectado ao *"else"* é realizada.

Exemplo: O algoritmo da Figura 34 verifica se um número é par ou ímpar.

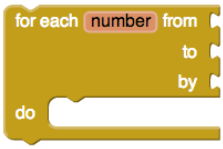
Figura 34 – Verificação se o número é par ou ímpar.



Este algoritmo é executado de maneira muito semelhante ao algoritmo anterior, porém, quando a condição verificada não é verdadeira, a ação definida em *Else* será executada. É possível observar na Figura 34, que se o resto da divisão de "TB_Numero" por 2 é igual a zero, então o *label* "LB_Par" receberá o texto "O número digitado é par". Senão, o *label* "LB_Par" receberá o texto "O número digitado é ímpar".



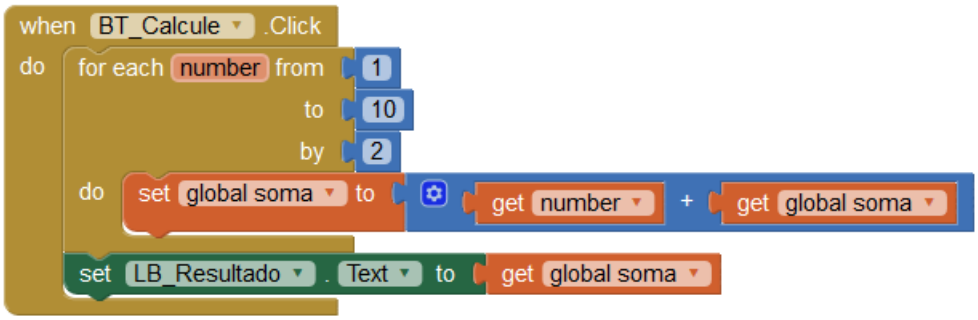
ElseIf: é possível utilizar a função modificadora do bloco *If* para adicionar um bloco *ElseIf*, criando condicionais em cascata.



Bloco *for*: conhecido como laço *for*, este bloco é utilizado para realizar uma determinada ação várias vezes, sendo que na primeira vez o item numérico vale *"from"*, e a cada ação o item numérico é incrementado em *"by"* unidades e as repetições vão até o item atingir o valor *"to"*.

O exemplo da Figura 35 efetua a soma dos números ímpares no intervalo entre 1 e 10.

Figura 35 – Exemplo de aplicação da função *for*.



Neste exmplo, o laço *For* opera da seguinte forma:

- Passo 1: A variável *"number"* recebe valor igual a 1 (pois foi definido em *from*), posteriormente salva a variável global "soma" como o resultado da adição da variável *"number"* (que atualmente é igual a 1) à variável global "soma", (que atualmente é igual a zero). Neste momento, a variável global soma é igual a 1;
- Passo 2: A variável *"number"* recebe valor igual a 3, (pois o incremento, definido em *by* é igual a 2) posteriormente salva a variável global "soma" como o resultado da adição da variável *"number"* (que atualmente é igual a 3) à variável global "soma", (que atualmente é igual a 1). Neste momento, a variável global soma é igual a 4;
- Passo 3: A variável *"number"* recebe valor igual a 5, posteriormente salva a variável global "soma" como o resultado da adição da variável *"number"* (que atualmente é igual a 5) à variável global "soma", (que atualmente é igual a 4). Neste momento, a variável global soma é igual a 9;
- Passo 4: A variável *"number"* recebe valor igual a 7, posteriormente salva a variável global "soma" como o resultado da adição da variável *"number"* (que atualmente é igual a 7) à variável global "soma", (que atualmente é igual a 9). Neste momento, a variável global soma é igual a 16;
- Passo 5: A variável *"number"* recebe valor igual a 9, posteriormente salva a variável global "soma" como o resultado da adição da variável *"number"* (que atualmente é igual a 9) à variável global "soma", (que atualmente é igual a 16). Neste momento, a variável global soma é igual a 25. Após realizar o passo 5, o bloco *for* encerra sua execução, pois a variável *number* receberia valor igual a 11, superior ao que foi definido na entrada *to*;

1.7.3 BLOCOS LÓGICOS

Os principais blocos lógicos utilizados nesse trabalho são: *true*, *false not*, *and* e *or*.

Blocos *true* e *false*: os blocos *true* e *false* são respectivamente, entradas verdadeira e falsa.

Bloco *not*: é uma porta inversora, ela retorna valor falso se a entrada é verdadeira e retorna o valor verdadeiro se a entrada for falsa.

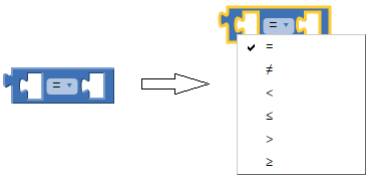
Bloco *and*: esse bloco analisa duas entradas e retorna valor verdadeiro somente se as duas entradas são verdadeiras. Vale salientar que uma entrada sem blocos conectados é considerada falsa.

Bloco *or*: esse bloco analisa duas entradas e retorna valor falso somente se as duas entradas são falsas. Vale salientar que uma entrada sem blocos conectados é considerada falsa.

1.7.4 BLOCOS MATEMÁTICOS

São os blocos utilizados para realizar operações matemáticas ou utilizar objetos matemáticos, como números por exemplo. Os principais blocos matemáticos utilizados neste trabalho são:

Bloco 0: número básico, padronizado inicialmente como zero, porém, clicando duas vezes sobre o "0", é possível alterar seu valor para qualquer número. Observação: uma entrada que não possui nenhum bloco conectado tem como padrão o valor igual a zero.



Bloco = : compara seus dois números de entrada, caso sejam iguais, o bloco retorna verdadeiro e caso não o seja, retorna falso. Utilizando a ferramenta alterar função, é possível que o bloco transforme-se nos blocos: $\neq, <, \leqslant, >$ e $\geqslant$.

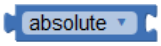
Bloco +: soma seus números de entrada, como o bloco é modificador, então é possível acrescentar mais entradas à ele.

Bloco -: subtrai o número da segunda entrada do número da primeira entrada.

Bloco x: multiplica os números das entradas, como o bloco é modificador, é possível aumentar o número de entradas.

Bloco /: divide o número na primeira entrada pelo número na segunda entrada.

Bloco $^\wedge$: eleva o número da primeira entrada ao número da segunda entrada.



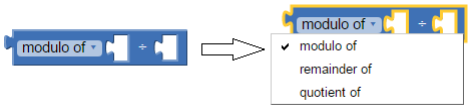
Bloco *absolute*: retorna o valor absoluto de um número.

Exemplo: O exemplo da Figura 36 a seguir verificará se a primeira entrada é maior do que a segunda entrada.

Figura 36 – Exemplo de operação matemática.

Na primeira entrada, tem-se o produto dos números 2, 3 e 0 (pois não há nenhum bloco conectado em um dos fatores) e na segunda entrada tem-se o número 4 elevado à potência 0. Como o produto da primeira entrada é igual a zero e a potência da segunda entrada resulta em 1, a comparação é falsa logo, a verificação é falsa.

Bloco *floor*: retorna o maior número inteiro menor do que ou igual à expressão numérica dada.



Bloco *modulo of*: retorna o resto da divisão do número na primeira entrada pelo número na segunda entrada. É possível alterar a função do bloco *modulo of* para *remainder of* e *quotient of*.

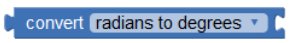
Este bloco aceita números negativos, porém, como neste trabalho isso não ocorrerá, tal caso não será abordado.

Exemplo: O exemplo da Figura 37 calcula o resto da divisão do número da primeira entrada pelo número da segunda entrada.

Figura 37 – *Modulo of*.

Como $14 = 11 \cdot 1 + 3$, então o bloco *modulo of* retornará o valor numérico igual a 3, pois é o resto dessa divisão. É possível extrair ainda, que 14 tem o mesmo módulo que 3 na divisão por 11, quando isso ocorre, diz-se que 14 é congruente a 3 módulo 11, porém, o conceito matemático que envolve congruência de módulos, será abordado posteriormente.

Blocos seno, cosseno e tangente: retornam os valores de seno, cosseno e tangente dos ângulos dados (em graus).

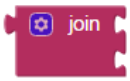


Bloco *Convert*: converte um ângulo dado em radiano para graus.

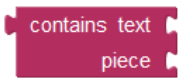
1.7.5 BLOCOS DE TEXTOS

Os blocos de texto mais utilizados neste trabalho são os blocos *String* e *Join*, descritos a seguir:

Bloco *String*: bloco de texto, que pode conter diversos caracteres, como números, letras e caracteres especiais. O *MIT App Inventor 2* sempre o considerará como um objeto de texto.



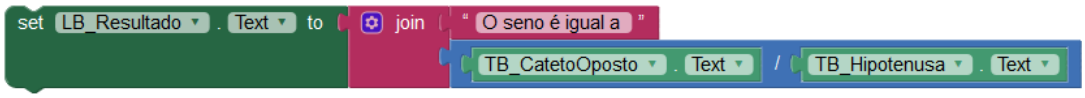
Bloco *Join*: Bloco modificador, que junta todas as entradas em uma um só bloco *String*. Caso não haja elementos na entrada, retorna uma *string* vazia.



Bloco *Contains*: Retorna valor verdadeiro se a entrada conectada ao *piece* está presente na entrada conectada ao *text* e retorna falso em caso contrário.

No exemplo da Figura 38 a seguir, o bloco *join* conecta suas duas entradas, sendo que a primeira entrada é o bloco *String*, definido com o texto "O seno é igual a" e a segunda entrada é o resultado da divisão do conteúdo do bloco "TB_CatetoOposto" pelo conteúdo do bloco "TB_Hipotenusa".

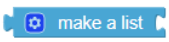
Figura 38 – Exemplo dos blocos de texto.



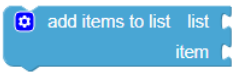
1.7.6 BLOCOS DE LISTAS

São os blocos utilizados para criar, editar e utilizar listas. Os principais blocos utilizados neste trabalho serão descritos a seguir:

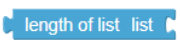
Bloco *Empty list*: cria uma lista sem nenhum elemento. O modificador desse bloco transforma-o no bloco *make a list*.



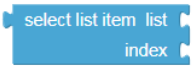
Bloco *Make a list*: esse bloco permite criar uma lista manualmente, a função modificadora desse bloco permite inserir mais itens à lista.



Bloco *Add itens*: esse bloco permite adicionar itens à uma lista, a função modificadora desse bloco permite inserir mais itens.



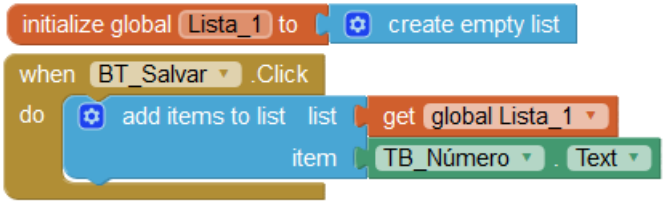
Bloco *Length List*: esse bloco fornece o tamanho da lista.



Bloco *Select item*: esse bloco permite selecionar determinado item da lista.

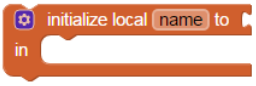
No exemplo da Figura 39 é criada uma lista vazia com o nome de "Lista_1", definida pela variável global "Lista_1". Quando o botão "BT_Salvar" é clicado, o texto contido na caixa de texto "TB_Número" é adicionado à lista.

Figura 39 – Exemplo utilizando blocos de listas.



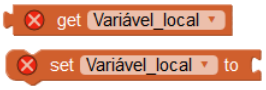
1.7.7 BLOCOS DE VARIÁVEIS

Bloco variável global: esse bloco é usado para criar variáveis globais e assume qualquer tipo de valor como argumento. As variáveis globais são independentes, pois podem ser utilizadas em diversos blocos simultaneamente e tem a possibilidade de permanecerem ativas durante todo o processamento. Tem *"name"* como nome padrão, mas é possível alterá-lo.



Bloco variável local: bloco modificador, similar às variáveis globais, porém, são dependentes, ou seja, somente podem ser declaradas dentro de outros blocos, e somente permanecem ativas durante a execução do bloco ao qual está inserida. Seu nome padrão, *"name"*, também pode ser alterado.

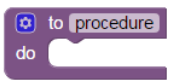
Valor inicial: os blocos de variáveis globais e locais tem que ter obrigatoriamente, um valor inicial.



Erro na variável local: como a variável só pode ser utilizada dentro de outros blocos, ao selecionar suas funções *get* e *set* (que serão explicadas posteriormente), é possível que apareça o círculo em vermelho com um "x", indicando erro. Isso ocorre porque a variável está sendo utilizada fora do bloco em que a variável local foi iniciada. Para que esse erro não aconteça, basta que os blocos referentes às variáveis locais sejam inseridos no interior das variáveis locais declaradas, como mostra a Figura 40.

Figura 40 – Corrigindo erro da variável local.

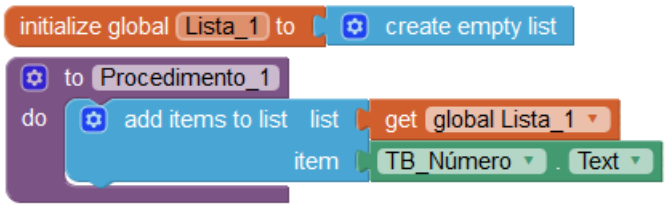
1.7.8 BLOCOS DE PROCEDIMENTOS



Bloco *Procedure*: o bloco *to procedure do* condensa o conjunto de blocos como um único bloco, que quando é chamado, realiza todas as ações a ele designado.

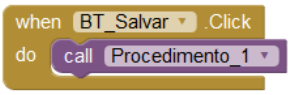
No exemplo da Figura 41, o “Procedimento_1” realiza a operação de adicionar itens à lista, como no exemplo da Figura 39.

Figura 41 – Exemplo utilizando blocos de procedimento.



Ao conectar o bloco *“call Procedimento_1”* ao bloco do botão “BT_Salvar”, como na Figura 42, quando o botão “BT_Salvar” for clicado, o comando definido em “Procedimento_1” será executado.

Figura 42 – Exemplo utilizando blocos de procedimento.

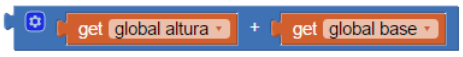


1.7.9 BLOCOS COMUNS ENTRE CONJUNTOS DE BLOCOS

Há alguns blocos que são comuns entre conjuntos de blocos, são eles:

Get: Esse bloco faz a comunicação entre dois blocos, por exemplo, na Figura 43, o bloco matemático “+” está recebendo os dados armazenados nas variáveis globais “altura” e “base” e somando-as. O bloco *get* pode ser referido como função *get* durante o desenvolvimento deste trabalho.

Figura 43 – Bloco *Get*.



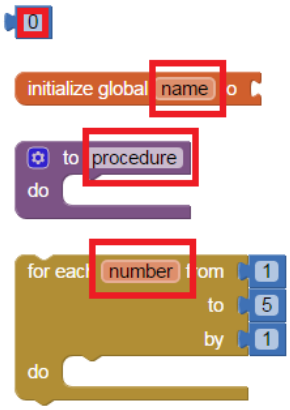
Set: Esse bloco permite estabelecer dados em um determinado bloco. Por exemplo, na Figura 44 , o “*label*” *“LB_resultado”* recebe informações quanto ao seu texto, que será definido pela soma das variáveis altura e base e recebe também informações quanto a cor de fundo que apresentará, que será a cor verde. O bloco *set* pode ser referido como função *set* durante o desenvolvimento deste trabalho.

Figura 44 – Bloco *Set*.

1.8 FERRAMENTAS ÚTEIS

Há determinadas ferramentas do *MIT App Inventor 2* que serão utilizadas durante o desenvolvimento dos aplicativos. As principais ferramentas serão descritas a seguir:

Figura 45 – Alterar nomes e valores.



Alterar nomes e valores: é possível alterar nomes e valores de alguns blocos. Os dados alteráveis dos blocos estão dentro de caixas de textos. Na Figura 45 encontram-se alguns exemplos destes blocos e em destaque, as caixas de textos com as informações que podem ser alteradas.

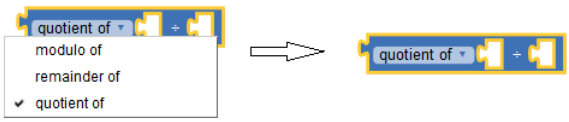
Atalhos do teclado: ações como copiar, colar, recortar e deletar podem ser realizadas através do teclado.

Figura 46 – Alterar função.

Alterar função do bloco: é possível alterar as funções de alguns blocos, esses blocos tem um pequeno triângulo indicando que há mais funções disponíveis, como na Figura 46, para exemplificar será utilizado o bloco *"modulo of"*, disponível no conjunto de blocos *"Math"*.

Clicando no local indicado na Figura 46, serão mostradas as funções disponíveis, basta então clicar na função desejada que a finalidade do bloco será alterada, como pode ser observado na Figura 47, o bloco *"modulo of"* teve sua função alterada para *"quotient of"*, ou seja, ao invés do bloco fornecer o valor do módulo entre as duas entradas, o bloco fornecerá o valor do quociente da divisão do valor na primeira entrada pelo valor na segunda entrada.

Figura 47 – Aba de alteração de função.



Inline* e *External Inputs: altera o formato do bloco, mais precisamente, altera a forma em que outros blocos encaixam-se no bloco em que será utilizada a ferramenta. Para utilizá-la deve-se clicar com o botão direito do *mouse* sobre o bloco e selecionar a opção *Inline/External Input* (dependendo do modo que o bloco está configurado), vale ressaltar que a função do bloco não é alterada. A forma de utilizar a ferramenta *External Inputs* é ilustrada na Figura 48. A ferramenta *Inline Inputs* é análoga.

Figura 48 – Inline External Inputs.

Collapse Block: condensa um bloco ou conjunto de blocos em um único bloco, para utilizar essa ferramenta, deve-se clicar com o botão direito do *mouse* sobre o bloco e selecionar a opção *"Collapse Block"*, como na Figura 49.

Figura 49 – Colapse Block.

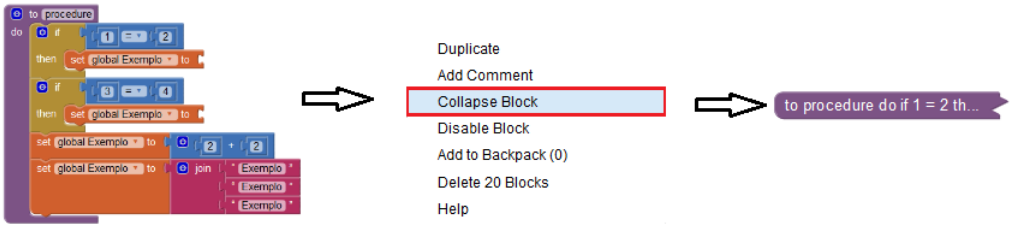
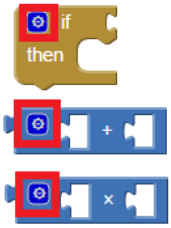


Figura 50 – *Mutator* ou Modificador.



Mutator: disponível em alguns blocos, o *mutator*, ou modificador, permite expandir, reduzir e até mesmo alterar as funções de um determinado bloco. Um bloco modificador é identificado por uma pequena engrenagem em um fundo azul, destacados por um contorno em vermelho, na Figura 50. São exemplos de blocos modificadores: "If", "+" e "×".

Figura 51 – Funções *Get* e *Set*.

Atalho para *Get* e *Set*: disponível nos blocos de variáveis, é possível criar blocos *Get* e *Set* movendo o cursor do *mouse* para o nome de alguma variável, como no exemplo da Figura 51.

2 APLICATIVO QUE DIA FOI

Eventualmente, em programas de televisão aparecem pessoas que supostamente possuem “habilidades especiais", por exemplo, que sabem qual foi o dia da semana de uma data aleatoriamente escolhida. Essas pessoas não decoraram todas essas datas, portanto é razoável supor que elas seguem um determinado algoritmo que torne possível descobrir o dia da semana de desejado data.

Neste capítulo, propomos como uma alternativa de aprendizagem baseada em projetos o desenvolvimento de um aplicativo, intitulado Calendário, na plataforma do *MIT App Inventor 2*, cujo objetivo é verificar o dia da semana correspondente a uma data escolhida.

Este projeto envolve principalmente os conceitos matemáticos de congruência e divisibilidade, além das habilidades inerentes a programação.

2.1 MOTIVAÇÃO, DISCUSSÃO DO PROBLEMA E OBJETIVOS

Saber o dia da semana correspondente a uma data importante, como fatos históricos, aniversários e feriados é algo interessante e embora pareça muito trabalhoso é perfeitamente possível. Existem inúmeras páginas da web e aplicativos que nos trazem essa informação, porém, não é tão óbvio o método empregado para se chegar a esse resultado.

Esse capítulo apresenta um método para determinar o dia da semana correspondente a uma data escolhida e propõe um aplicativo que o aluno é capaz de desenvolver, instalar em seu aparelho celular e ainda compartilhar esse aplicativo.

Espera-se que com essa atividade, o aluno aprofunde seus conhecimentos em congruência e divisibilidade, perceba a presença da matemática em diferentes aplicativos além de desenvolver as habilidades inerentes à matemática e programação.

Acreditamos que a complexidade da construção desse aplicativo é adequada a estudantes do ensino fundamental. Além de congruência e divisibilidade outros conhecimentos matemáticos podem surgir ao longo do trabalho, assim como ideias para outros aplicativos.

2.2 MATEMÁTICA DO APLICATIVO QUE DIA FOI

A caracterização de um ano como bissexto e a determinação do dia da semana, utilizam os conceitos matemáticos de divisibilidade e congruência. Vamos rever os principais resultados sobre esses conceitos, pertinentes ao desenvolvimento do aplicativo *Que dia foi*.

Definição 2.1 (Congruência)**.** *Seja m um número natural. Diremos que dois números inteiros a e b são congruentes módulo m se os restos de sua divisão euclidiana por m são iguais. Quando*

os inteiros a e b são congruentes módulo m, escreve-se:

$$a \equiv b \mod m.$$

Quando a relação $a \equiv b \mod m$ for falsa, diremos que a e b não são congruentes, ou que são incongruentes, módulo m. Escrevemos, nesse caso, $a \not\equiv b \mod m$. (HEFEZ, 2014)

Por exemplo, $21 \equiv 13 \mod 2$, já que os restos da divisão de 21 e de 13 por 2 são iguais a 1 e tem-se que $14 \not\equiv 5 \mod 4$, pois o resto das divisões de 14 e 5 por 4 são diferentes, a saber, os restos são respectivamente, iguais a 2 e 1.

Teorema 2.2 (Divisão Euclidiana)**.** *Sejam a e b dois números inteiros com $b \neq 0$. Existem dois únicos números inteiros q e r tais que a=bq+r, com $0 \leqslant r < |b|$.*

Demonstração. Considere o conjunto:

$$S = \{x = a - by; y \in \mathbb{Z}\} \cap (\mathbb{N} \cup 0).$$

Existência: Pela propriedade arquimediana, existe $n \in \mathbb{Z}$ tal que $n(-b) > -a$, logo $a - nb > 0$, o que mostra que S é não vazio. O conjunto S é limitado inferiormente por 0, logo, pelo Princípio da Boa Ordenação, temos que S possui um menor elemento r. Suponhamos então que $r = a - bq$. Sabemos que $r \geq 0$. Vamos mostrar que $r < |b|$. Suponhamos por absurdo, que $r \geq |b|$. Portanto, existe $s \in \mathbb{N} \cup \{0\}$ tal que $r = |b| + s$, logo $0 \leqslant s < r$. Mas isso contradiz o fato de r ser o menor elemento de S, pois $s = a - b(q \pm 1) \in S$, com $s < r$.

Unicidade: Suponha que $a = bq + r = bq' + r'$, onde $q, q', r, r' \in \mathbb{Z}, 0 \leq r < |b| e 0 \leq r' < |b|$. Assim, temos que $-|b| < -r \leq r' - r \leq r' < |b|$. Logo, $|r' - r| < |b|$. Por outro lado, $b(q - q') = r' - r$, o que implica que:

$$|b||q - q'| = |r' - r| < |b|,$$

o que só é possível se $q = q'$ e consequentemente, $r = r'$. (HEFEZ, 2014) □

Definição 2.3 (Divisibilidade)**.** *Dados dois números inteiros a e b, diremos que a divide b, escrevendo $a|b$, quando existir $c \in \mathbb{N}$ tal que $b = ca$. Nesse caso, diremos também que a é um divisor ou um fator de b ou, ainda que b é um múltiplo de a ou que b é divisível por a. (HEFEZ, 2014)*

Para o aplicativo *Que Dia Foi*, serão utilizado critérios de divisibilidade por 4, 7, 100 e 400. Para que um número seja divisível por 4 basta que seus dois últimos dígitos formem um múltiplo de 4 e para que seja divisível por 100 os dois últimos digítos do número devem ser 0. Critérios de divisibilidade pelos números 4 e 100 são simples e não os demonstraremos. Neste trabalho, vamos demonstrar os critérios de divisibilidade por 7 e por 400.

Para o critério de divisibilidade por 400, será utilizada a Proposição 2.4 afim de decompor o número a ser analisado em duas partes.

Proposição 2.4. *Se $a, b, c, d \in \mathbb{Z}$, então $a|b$ e $c|d \Rightarrow ac|bd$.*

Demonstração. Se $a|b$ e $c|d$, então $\exists f, g \in \mathbb{Z}$ então, $b = fa$ e $d = gc$. Portanto, $bd = (fg)(ac)$, logo, $ac|bd$. (HEFEZ, 2014) □

Desse modo, suponha um número $a = a_k a_{k-1} \dots a_1 a_0$, este será divisível por 100 somente se os algarismos a_1 e a_0 forem iguais a 0, logo, só será necessário verificar se o número $a' = a_k a_{k-1} \dots a_3 a_2$ é divisível por 4, ou seja, basta que $a_1 = a_0 = 0$ e $a_3 a_2 = 4q$, com $q \in \mathbb{Z}$.

A seguir, será demonstrado um critério de divisibilidade por 7, que consiste em multiplicar por 2 o algarismo das unidades do número e então subtrair este valor do número inicial sem o algarismo das unidades. Caso esse número seja divisível por 7, então o número original também o é.

Proposição 2.5 (Critério de divisibilidade por 7)**.** *Um número inteiro $a = a_k a_{k-1} \dots a_1 a_0$ é divisível por 7, se e somente se, o número inteiro b=$a_k a_{k-1} \dots a_2 a_1 - 2a_0$ é divisível por 7.*

Demonstração. Sabe-se que é possível representar o número $a = a_k a_{k-1} \dots a_1 a_0$ através do somatório $\sum_{i=0}^{k} a_i 10^i$, analogamente, é possível representar o número b=$a_k a_{k-1} \dots a_2 a_1 - 2a_0$ como $\sum_{i=1}^{k} a_i 10^{i-1} - 2a_0$.

Assim, sendo, suponha inicialmente, que $a = \sum_{i=0}^{k} a_i 10^i = 7q$ com $q \in \mathbb{Z}$. Então,

$$a = \sum_{i=1}^{k} a_i 10^i + a_0 = 7q \Rightarrow 2\sum_{i=1}^{k} a_i 10^i + 2a_0 = 2 \cdot 7q \Rightarrow -2a_0 = 2\sum_{i=1}^{k} a_i 10^i - 2 \cdot 7q. \quad (2.1)$$

Será provado agora, que $\sum_{i=1}^{k} a_i 10^{i-1} - 2a_0$ é divisível por 7:

Substituindo a Equação 2.1 em $\sum_{i=1}^{k} a_i 10^{i-1} - 2a_0$, tem-se que:

$$\begin{aligned}
\sum_{i=1}^{k} a_i 10^{i-1} - 2a_0 &= \sum_{i=1}^{k} a_i 10^{i-1} + 2\sum_{i=1}^{k} a_i 10^i - 2 \cdot 7q \\
&= \sum_{i=1}^{k} a_i 10^{i-1} + 2 \cdot 10 \sum_{i=1}^{k} a_i 10^{i-1} - 2 \cdot 7q \\
&= (1 + 20) \sum_{i=1}^{k} a_i 10^{i-1} - 2 \cdot 7q \\
&= 7(3 \sum_{i=1}^{k} a_i 10^{i-1} - 2q).
\end{aligned}$$

O que prova que $\sum_{i=1}^{k} a_i 10^{i-1} - 2a_0$ é divisível por 7.

Reciprocamente, suponha que $\sum_{i=1}^{k} a_i 10^{i-1} - 2a_0 = 7p$. Multiplicando ambos os lados por 10, tem-se que:

$$\sum_{i=1}^{k} a_i 10^i - 20a_0 = 7 \cdot 10p \Rightarrow \sum_{i=1}^{k} a_i 10^i + a_0 - (20+1)a_0 = 7 \cdot 10p.$$

Logo,

$$\sum_{i=1}^{k} a_i 10^i + a_0 = 21a_0 + 7 \cdot 10p = 7(3a_0 + 10p).$$

O que prova que $\sum_{i=1}^{k} a_i 10^i + a_0 = \sum_{i=0}^{k} a_i 10^i$ é divisível por 7. (FONSECA; ALMADA, 2013) □

É possível observar que esse critério pode ser utilizado sucessivamente até que se obtenha um número fácil de ser analisado quanto a sua divisibilidade por 7.

Critérios de divisibilidade para outros números podem ser encontrados em (FONSECA; ALMADA, 2013).

2.3 IMPLEMENTAÇÃO DO APLICATIVO QUE DIA FOI

É preciso determinar o dia da semana correspondente a uma data. Nos baseamos na função "*DIA.DA.SEMANA*" do *Microsoft Excel* (MICROSFOT CORPORATION, 2017, Acesso em: 20 jun. 2017) e no algoritmo disponível em (SOUSA, 2013, Acesso em: 20 jun. 2017) para determinar o dia da semana. A estratégia adotada para encontrar essa data baseia-se na contagem de dias transcorridos a partir de uma data de referência.

Por exemplo, suponha que o dia da semana de uma determinada data corresponde a uma terça-feira. O dia da semana correspondente a 100 dias após essa data será uma quinta feira. Isso pode ser verificado facilmente baseando-se no fato dos dias da semana terem um ciclo de 7 dias, basta então contar quantas vezes este ciclo ocorreu e para isso, utiliza-se o conceito matemático da divisão Euclidiana. Como $100 = 14 \cdot 7 + 2$, passaram-se 14 ciclos e mais dois dias.

Desse modo, verifica-se que o número x relevante para a contagem dos dias do aplicativo, é o resto da divisão do número de dias corridos por 7, ou seja, desejamos um x tal que:

$$x \equiv \text{número de dias corridos} \mod 7.$$

A data de referência utilizada para desenvolver o aplicativo foi um suposto dia zero de janeiro do ano zero. Essa data foi escolhida afim de facilitar a contagem de dias decorridos, para que a data escolhida seja sempre posterior à data de referência.

A contagem de dias transcorridos torna-se um contratempo conforme as datas afastam-se da data de referência, pois além do elevado número de dias, há ainda as diferentes quantidades de dias que cada mês possui, levar em conta também os anos bissextos e, em caso de ano bissexto, é preciso considerar que as datas anteriores a 1º de março não sofre a influência do acréscimo do dia 29 de fevereiro.

O método adotado para deslindar esses contratempos foi resolver cada um deles separadamente, utilizando os conceitos matemáticos de divisibilidade e congruência em diversas etapas.

Há determinados termos criados para a compreensão e implementação do aplicativo que serão utilizados diversas vezes. Para melhor entendimento, estes termos serão definidos a seguir:

- **Dia do ano**: Entende-se por dia do ano, uma data definida por um dia e um mês, por exemplo, os dias 15 de abril, 4 de dezembro e 12 de setembro serão chamados dia do ano.
- **Dia do mês**: Entende-se por dia do mês, um número compreendido entre 1 e 31, a depender do mês de que se trata o dia, pois há meses com 28, 29, 30 e 31 dias.
- **Dia da semana**: Entende-se por dia da semana um dia pertencente ao ciclo de domingo a sábado.
- **Dias à frente**: Este termo será sempre referente aos dias da semana, por exemplo, o dia da semana que é 3 dias à frente da segunda-feira, é a quinta-feira. Se a contagem de dias à frente chegar ao sábado, como os dias da semana são cíclicos, a contagem continua no domingo. Por exemplo, o dia da semana que é 5 dias à frente da quinta-feira é a terça-feira, o dia da semana que é 2 dias à frente do sábado é a segunda-feira.

O dia da semana que ocorreu uma data escolhida pode ser determinada por meio da Equação 2.2.

$$\text{O dia da semana} \equiv (\text{Ano} + \text{Chave do mês} + \text{Bissextos} + \text{Dia escolhido})(\mod 7). \quad (2.2)$$

De acordo com o Teorema 2.2, essa equação determina um número entre 0 e 6 que posteriormente será relacionado a um dia da semana. O modo de desenvolver o aplicativo *Que Dia Foi* e cada item dessa equação serão descritos a seguir.

2.3.1 CONFIGURAÇÃO DA TELA DE DESIGNER

Em nossa implementação os blocos da tela de *Designer* foram posicionados como mostrado na Figura 52.

Figura 52 – Viewer do aplicativo.

Note que o bloco “LB_Resultado” não aparece na Figura 52, isso ocorre porque ele foi configurado para aparecer somente na apresentação do resultado.

Para que os blocos da tela de *Designer* fiquem posicionados dessa maneira, serão necessários os seguintes blocos e configurações:

- ***Button***: Um botão, renomeá-lo como “BT_Verifique” e alterar a propriedade *Width* para *Fill parent*;
- ***Label***: Quatro *labels*, renomeá-los como “LB_Mês”, “LB_Dia”, “LB_Ano” e “LB_Resultado” e modificar as propriedades *Text* de “LB_Mês”, “LB_Dia” e “LB_Ano” respectivamente para “Selecione um mês”, “Selecione um dia” e “Digite um ano”. As propriedades do *label* “LB_Resultado” não devem ser alteradas;
- **Caixa de texto**: Uma caixa de texto, renomeá-la como “TB_Ano”, alterar a propriedade *Hint* para “Digite um ano” e marcar a opção *NumbersOnly* para que seja possível apenas inserir números;
- ***Spinner***: Dois *spinners*, renomeando-os como “SP_Mês” e “SP_Dia”, alterar a propriedade *Prompt* respectivamente para “Selecione um mês” e “Selecione um dia”, inserir na propriedade *ElementsFromString* do bloco “SP_Mês” o texto: “Selecione, Janeiro, Fevereiro, Março, Abril, Maio, Junho, Julho, Agosto, Setembro, Outubro, Novembro, Dezembro”. A opção de utilizar *Spinners* ao invés de caixas de texto para as entradas de dia e mês foi feita para evitar que o usuário insira datas inválidas, por exemplo, digitando

um dia que não exista em determinado mês, ou ainda, erros ao digitar o nome do mês escolhido;

- ***Notifier***: Um bloco *Notifier*;
- **Blocos de *Layout***: Um bloco *TableArrangement*, alterar o número de linhas para 3 e alterar a propriedade *Width* para *Fill parent*;

2.3.2 ANO SELECIONADO

Um ano não bissexto tem 365 dias (os anos bissextos serão considerados posteriormente), ao dividir 365 por 7, obtém-se resto igual a 1. Por isso, se em um ano não bissexto o Natal ocorre em uma terça-feira, no ano seguinte, o Natal ocorrerá na quarta feira. É como se cada dia do ano seguinte fosse deslocado um dia da semana para frente. Assim, se todos os anos fossem não bissextos, do ano "zero" a um ano x, os dias da semana seriam deslocados x dias para frente.

A caixa de texto "TB_Ano" servirá como entrada para o ano escolhido.

2.3.3 CHAVE DO MÊS

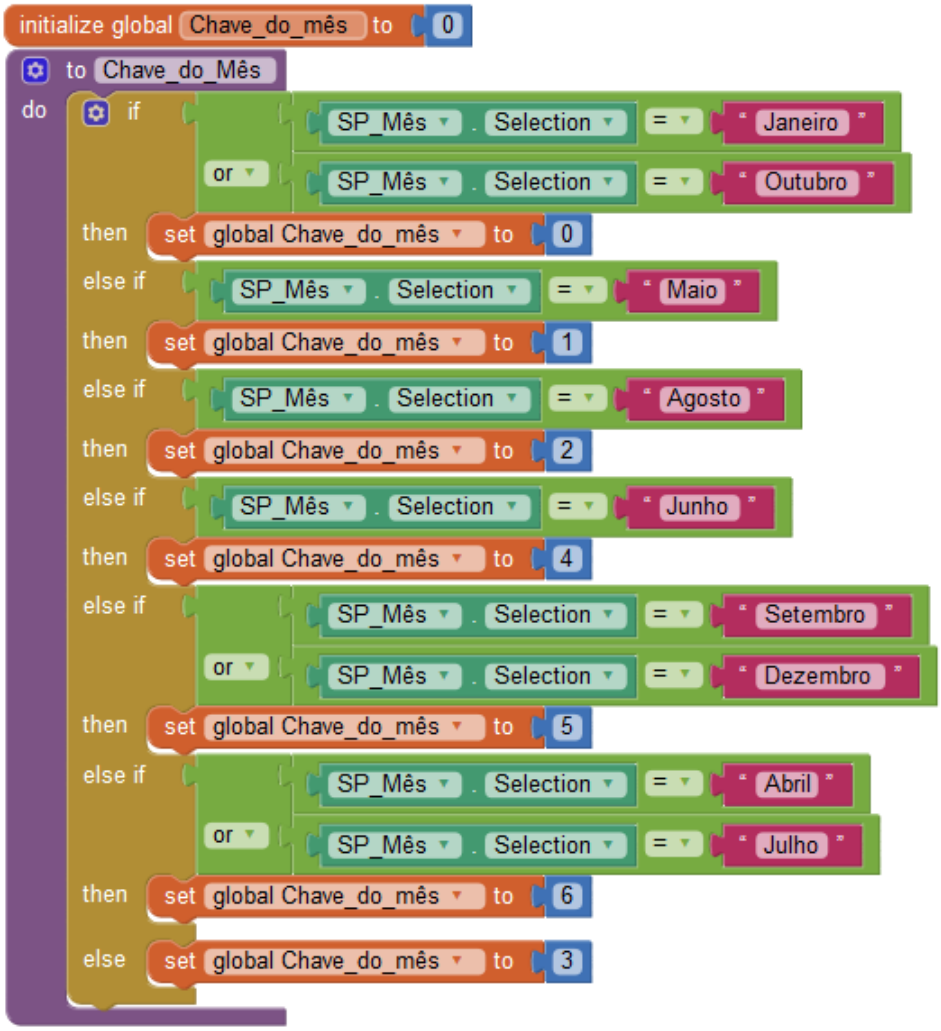
A cada mês do ano, entre os meses de janeiro e dezembro, será associado um número, chamado chave do mês. Essa chave é definida de acordo com o deslocamento dos dias da semana relativos a um determinado mês de referência, de acordo com o seguinte raciocínio: tomamos como referência o mês de janeiro, que recebe chave do mês igual a zero, temos que a chave do mês de fevereiro é igual a 3, pois o mês de janeiro tem 31 dias e como 31 deixa resto igual a 3 na divisão por 7, tomando um dia x de janeiro, o mesmo dia do mês em fevereiro, se houver, será três dias da semana a frente.

Por exemplo, suponha que em determinado ano, o dia 11 de janeiro foi uma terça-feira, o dia 11 de fevereiro então será três dias da semana à frente da terça-feira, ou seja, uma sexta-feira.

Fevereiro por sua vez tem 28 dias, que deixa resto igual a zero na divisão por 7, desse modo, março não tem seus dias deslocados em relação a fevereiro. Por exemplo, o dia 11 de fevereiro que no exemplo citado anteriormente foi uma sexta-feira, então o dia 11 de março será também uma sexta-feira. Desse modo, tem-se que a chave do mês de março também é igual a 3. Prosseguindo com este raciocínio, tem-se que a chave do mês de janeiro e outubro é igual a zero, maio é igual a 1, agosto é igual a 2, fevereiro, março e novembro são iguais a 3, junho é igual a 4, setembro e dezembro são iguais a 5, abril e julho são iguais a 6.

Precisamos definir a chave do mês de acordo com a escolha do usuário. O agrupamento de blocos referente ao algoritmo que utilizamos para definir a chave do mês é apresentado na Figura 53.

Figura 53 – Algoritmo chave do mês.



Para representar a chave do mês, foi criada uma variável, chamada “Chave_do_mês” com valor inicial igual a zero. Foi também criado um procedimento chamado “Chave_do_mês” afim de organizar a construção do aplicativo *Que Dia Foi*.

O algoritmo executa inicialmente a seguinte verificação: se o mês selecionado no *Spinner* é igual a janeiro ou outubro, então atribui-se valor igual a 0 à chave do mês.

Se o mês selecionado não for igual a janeiro ou outubro, é verificado se o mês selecionado é maio, se a verificação for verdadeira, então atribui-se valor igual a 1 à chave do mês e assim, sucessivamente.

Note que se todas as verificações forem falsas, isto é, se o mês selecionado não for igual a janeiro, outubro, maio, agosto, junho, setembro, dezembro, abril ou julho, então o mês escolhido só pode ter sido fevereiro, março ou novembro, o que deve atribuir valor igual a 3 à chave do mês.

2.3.4 BISSEXTOS

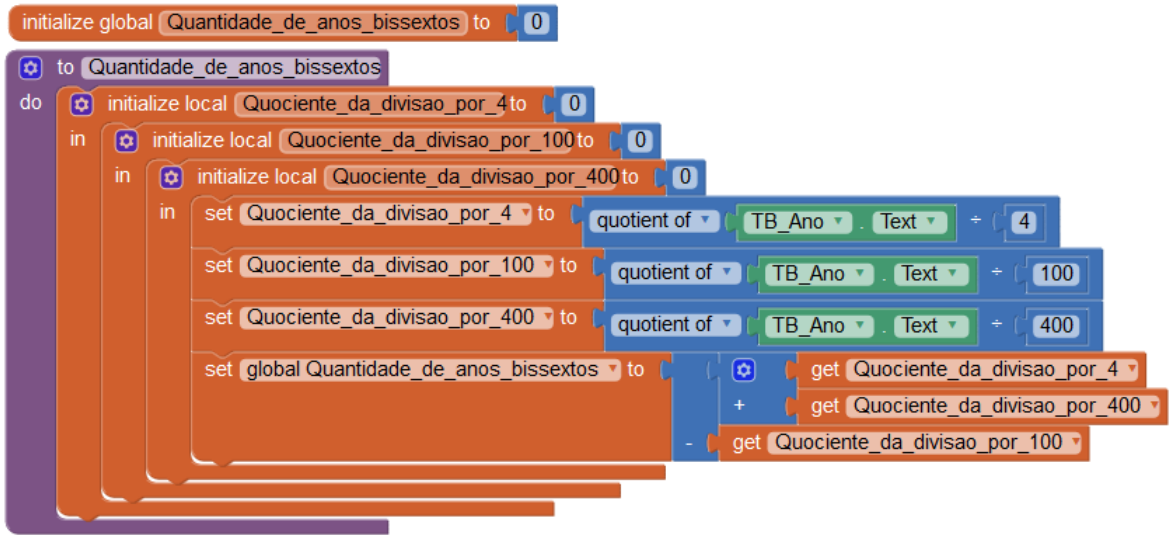
Para determinar a chaves do mês, supôs-se que os anos eram todos não-bissextos. Nesta seção será desenvolvido um algoritmo para incluir os anos bissextos. A diferença de um ano bissexto para um ano não-bissexto é o acréscimo do dia 29 de fevereiro, o que consequentemente desloca um dia da semana à frente a cada ano bissexto que ocorre. Desse modo é necessário realizar a contagem de quantos anos bissextos ocorreram do ano "zero" ao ano escolhido afim de incluir esse deslocamento.

Um ano é bissexto se atender a uma das duas condições: a primeira condição é que o ano deve ser múltiplo de 4 e não simultaneamente múltiplo de 100. A segunda condição é que o ano deve ser múltiplo de 400. Deste modo, para contar quantos anos bissextos ocorreram entre o ano "zero" e um ano x, devemos considerar a quantidade de anos múltiplos de 4, múltiplos de 100 e múltiplos de 400 houve nesse período, ou seja, serão necessárias as partes inteiras dos quocientes das divisões do ano escolhido pelos números 4, 100 e 400. A Equação 2.3 corresponde ao cálculo do número de anos bissextos ocorreram.

$$\text{Quantidade de anos bissextos} = \text{Q4} - \text{Q100} + \text{Q400}. \tag{2.3}$$

Onde Q4, Q100 e Q400 são as partes inteiras dos quocientes das divisões do número correspondente ao ano escolhido por 4, 100 e 400. O algoritmo que realiza a contagem de anos bissextos pode ser observado na Figura 54.

Figura 54 – Algoritmo da contagem de anos bissextos.



Para representar a contagem de anos bissextos foi criada uma variável chamada "Quantidade_de_anos_bissextos", também foram criadas três variáveis locais para realizar os cálculos dos quocientes e para organizar a construção do aplicativo, foi criado um procedimento, chamado "Quantidade_de_anos_bissextos".

Esse algoritmo calcula o quociente da divisão do número correspondente ao ano escolhido pelo usuário pelos números 4, 100 e 400 e os salva em suas respectivas variáveis.

Posteriormente, o resultado da Equação 2.3 é atribuído à variável correspondente à contagem de anos bissextos. Por exemplo, seja o ano de 2017, temos que o quociente da divisão de 2017 pelos números 4, 100 e 400 são, respectivamente, 504, 20 e 5. Substituindo esses números na Equação 2.3, temos que o número de anos bissextos que ocorreram do ano zero até o ano de 2017, é igual a 489.

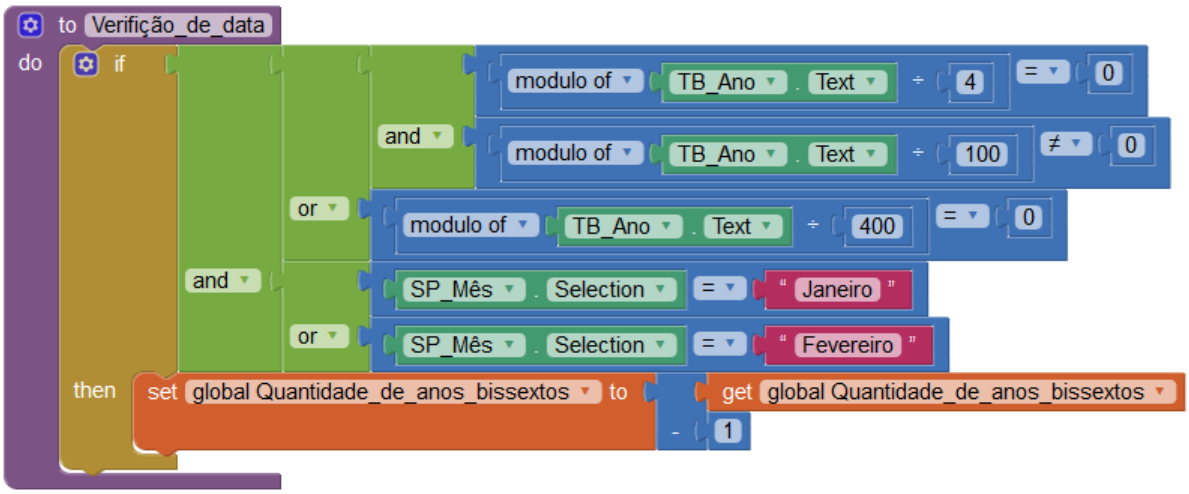
Para finalizar, é interessante utilizar a ferramenta *Collapse Block* sobre o procedimento, para reduzir o tamanho do algoritmo e organizar a tela de blocos.

2.3.5 O ANO SELECIONADO É BISSEXTO?

Se o ano selecionado for bissexto e a data escolhida pertencer aos meses de janeiro ou fevereiro, será necessário subtrair uma unidade da contagem de anos bissextos, pois ao realizar a contagem, o ano escolhido já está incluso e os dias anteriores a 1º de março não sofrem a influência do acréscimo do dia 29 de fevereiro.

O agrupamento de blocos correspondente ao algoritmo que verifica esse caso particular do aplicativo *Que Dia Foi* é apresentado na Figura 55.

Figura 55 – Verificação se um ano é bissexto.



Afim de organizar a construção do aplicativo *Que Dia Foi*, foi utilizado um procedimento, chamado de "Verificação_de_data" . Este algoritmo realiza a verificação se o ano é bissexto e se a escolha do mês foi igual a janeiro ou fevereiro e, em caso afirmativo, a contagem de anos bissextos é reduzida em uma unidade.

Como visto anteriormente, para que um ano seja bissexto ele deve ser múltiplo de 4 mas não simultaneamente múltiplo de 100 ou deve ser múltiplo de 400. Isso é o equivalente a dizer que o número correspondente ao ano deve ser congruente a zero módulo quatro e incongruente a zero módulo 100 ou o número correspondente ao ano deve ser congruente a zero módulo 400 e é exatamente essa verificação que pode ser observada na Figura 55.

Por exemplo, suponha que o usuário selecione um dia do mês de janeiro do ano de 2016. Como $2016 \equiv 0 \mod 4$ e $2016 \equiv 16 \mod 100 \Rightarrow 2016 \not\equiv 0 \mod 100$ e pelo mês escolhido ser janeiro, a verificação retorna valor verdadeiro, desse modo, a contagem de anos bissextos é subtraída em uma unidade.

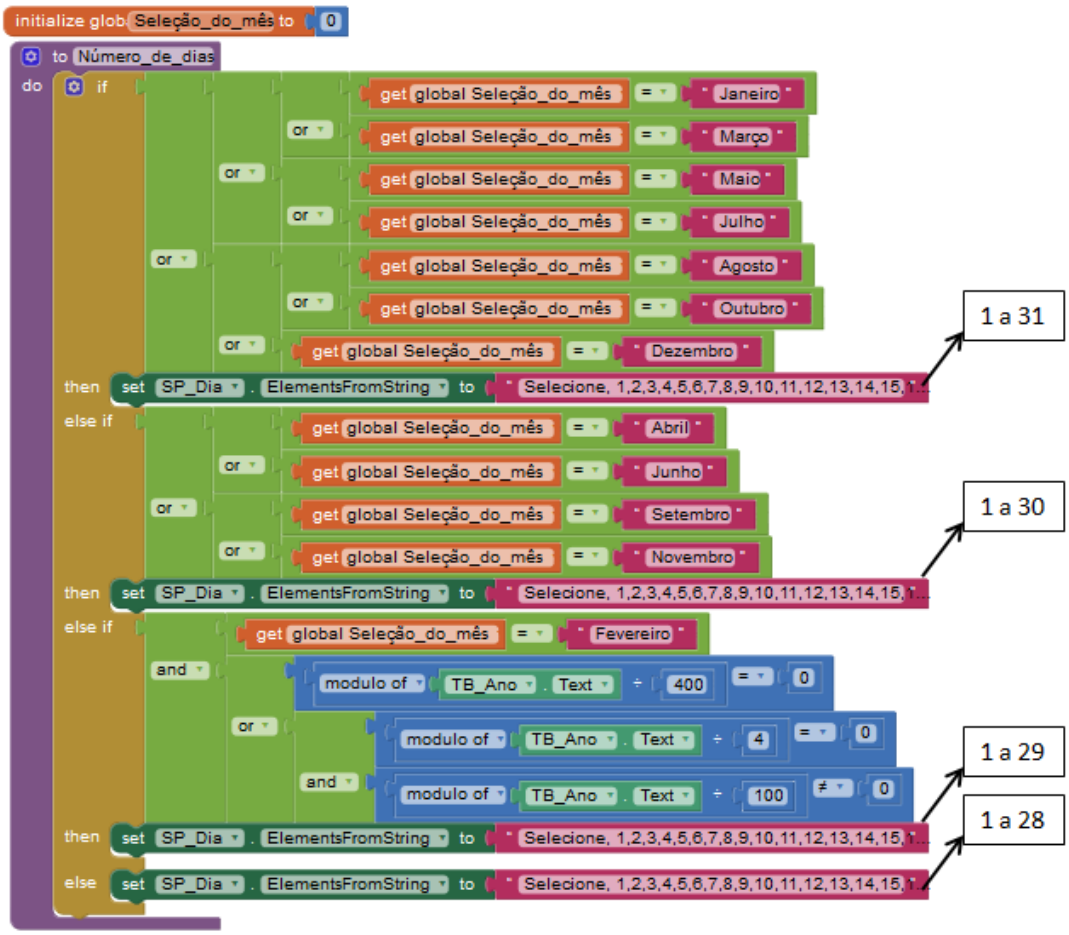
Para finalizar, é interessante condensar o procedimento utilizando *Collapse Block*.

2.3.6 NÚMERO DE DIAS

Os meses de janeiro, março, maio, julho, agosto, outubro e dezembro tem 31 dias. Os meses de abril, junho, setembro e novembro tem 30 dias e o mês de fevereiro pode ter 28 ou 29 dias condicionado à escolha do ano. Desse modo, é preciso configurar o *Spinner* correspondente à seleção do dia de acordo com o mês e também ano escolhido.

O algoritmo que realiza essa configuração é observado na Figura 56.

Figura 56 – Configurando *Spinner* para o número de dias.



Foi criada uma variável, chamada “Seleção_do_mês”, que representará a escolha do mês, foi criado também um procedimento, chamado “Número_de_dias”.

Esse algoritmo executa três verificações, na primeira é analisado se o mês escolhido é igual a janeiro, março, maio, julho, agosto, outubro ou dezembro caso essa verificação seja verdadeira, o *Spinner* "SP_Dia" recebe os números de 1 a 31, mas se a verificação for falsa, é realizada a segunda aferição.

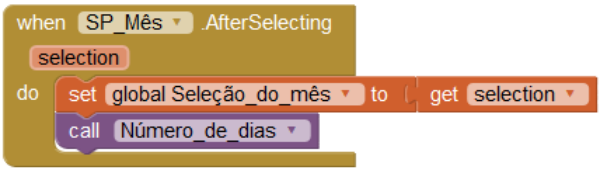
A segunda verificação analisa se o mês escolhido é igual a abril, junho, setembro ou novembro e caso essa verificação seja verdadeira, o *Spinner* "SP_Dia" recebe os números de 1 a 30, se a verificação for falsa, é realizada a terceira averiguação.

A terceira verificação analisa se o mês escolhido é igual a fevereiro e se o ano escolhido é bissexto. Caso essa verificação seja verdadeira, o *Spinner* "SP_Dia" recebe os números de 1 a 29 e se essa verificação for falsa, o *Spinner* "SP_Dia" recebe os números de 1 a 28.

Note que o primeiro elemento das *Strings* que configuram os dias iniciam-se com a palavra "Selecione", pois o primeiro elemento dos *Spinners* deve ser um elemento não selecionável.

Como citado anteriormente, os dias do *Spinner* "SP_Dia" devem ser configurados após o usuário selecionar um mês. Para que o aplicativo entenda que isso deve realizado, será necessário criar o algoritmo da Figura 57.

Figura 57 – Configurando o número de dias de acordo com o mês.



Esse algoritmo opera da seguinte forma: após o usuário selecionar o mês, a variável "Seleção_do_mês" é salva com essa informação e o procedimento "Número_de_dias", criado anteriormente, é executado.

2.3.7 CONFIGURAÇÃO DO DIA DA SEMANA

O calendário Juliano foi utilizado pela maioria dos países a partir do ano de 46 a.C. e foi substituído pelo calendário Gregoriano que é utilizado nos dias atuais. Somente a partir da adoção do calendário Gregoriano os algoritmos desenvolvidos neste capítulo fazem sentido, porém, optamos por permitir que o usuário possa verificar, a título de curiosidade, o dia da semana correspondente a uma data anterior à implementação do calendário Gregoriano. Seria interessante por exemplo, verificar o dia da semana em que o Brasil foi descoberto caso o calendário que utilizamos atualmente fosse também utilizado àquela época.

O calendário Gregoriano foi promulgado pelo Papa Gregorio XIII na data de 24 de fevereiro de 1582, sendo que o primeiro dia oficial deste calendário foi dia 15 de outubro de 1582, uma sexta-feira (CALENDARIO..., 2017, Acesso em: 21 jun. 2017). Para associar o

resultado da Equação 2.2 ao dia da semana, serão determinados a seguir, os dados necessários para calcular o valor da equação para essa data.

Ano selecionado: 1582.

Chave do mês: O mês é outubro, que tem chave igual a 0.

Bissextos: Como o quociente da divisão de 1582 por 4, 100 e 400 são respectivamente 395, 15 e 3, para saber a quantidade de anos bissextos basta realizar a operação $395 - 15 + 3 = 383$.

O ano selecionado é bissexto?: Como o ano escolhido não é bissexto, não interfere na contagem dos bissextos.

Dia selecionado: O dia selecionado é o dia 15.

Aplicando os dados na Equação 2.2, tem-se:

$$\begin{aligned} \text{O dia da semana foi} &\equiv \text{Ano} + \text{Chave do mês} + \text{Bissextos} + \text{Dia escolhido } (mod 7) \\ &\equiv 1582 + 0 + 383 + 15 \ (mod 7) \\ &\equiv 6 \ (mod 7) \end{aligned}$$

Desse modo, temos que se o resultado da equação é igual a 6 e como é sabido que o dia 15 de outubro de 1582 foi uma sexta-feira, o fato da Equação 2.2 ter resultado igual a 6, implica que o dia da semana corresponde a uma sexta-feira.

Assim sendo, os demais valores possíveis, isto é, 0, 1, 2, 3, 4 e 5 para o resultado da equação, correspondem respectivamente, a sábado, domingo, segunda-feira, terça-feira, quarta-feira e quinta-feira, isso ocorre pela maneira que foram realizados os cálculos para se determinar o dia da semana.

O algoritmo que relaciona os dias da semana com o resultado da Equação 2.2 é apresentado na Figura 58.

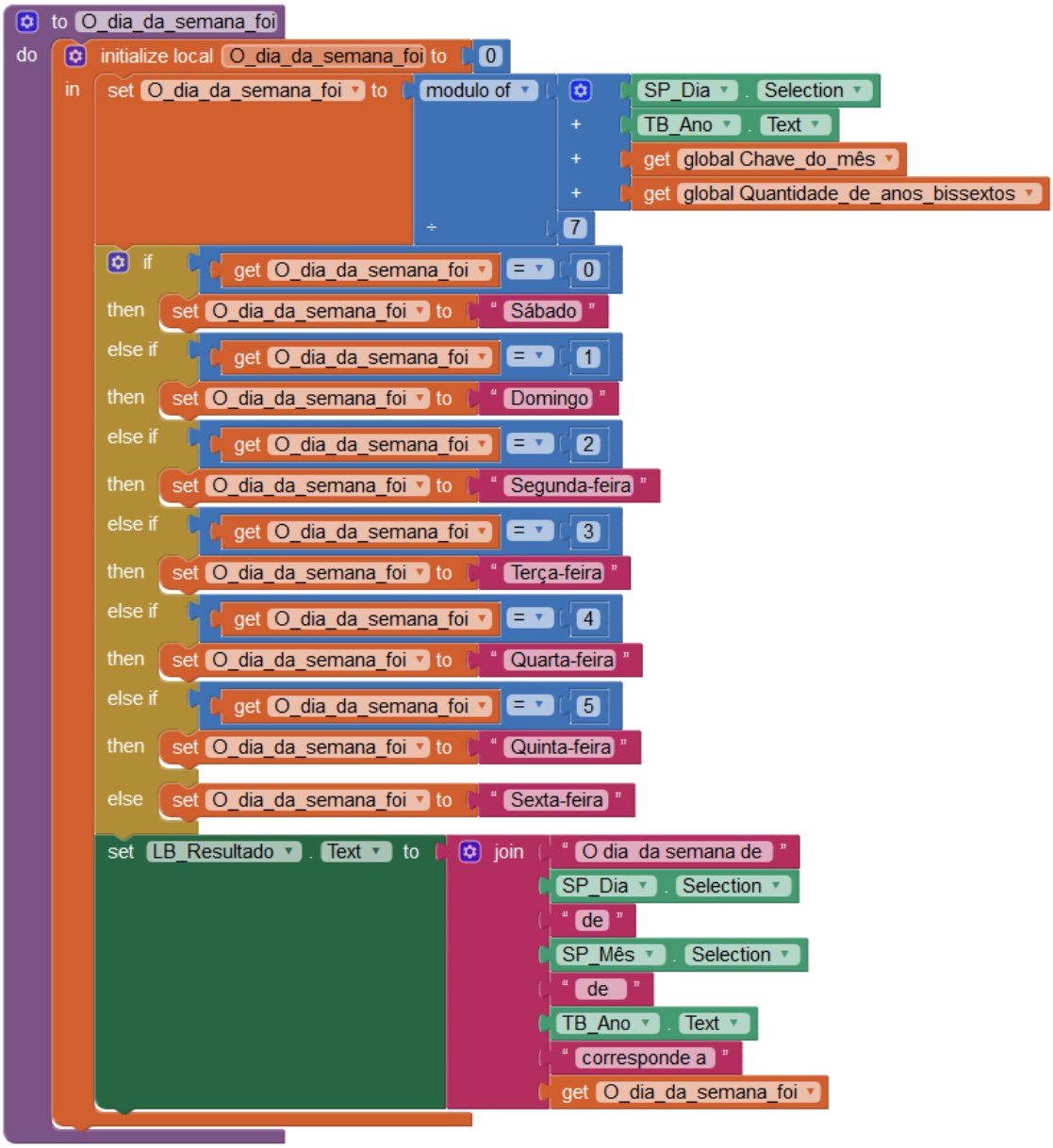
Foram criados um procedimento, chamado “O_dia_da_semana_foi” e uma variável local, de mesmo nome, que primeiramente salva o resultado da equação e posteriormente, salva o dia da semana correspondente.

Primeiramente é efetuado através do bloco *modulo of*, o cálculo da Equação 2.2 e então, é realizada a verificação do resultado da equação, salvando a variável “O_dia_da_semana_foi” com o dia da semana correspondente.

2.3.8 DEFININDO A FUNÇÃO DO BOTÃO VERIFIQUE

Quando o botão “BT_Verifique” for clicado, o aplicativo deve fornecer o dia da semana correspondente à data escolhida. Porém, é preciso considerar possíveis erros quando o

Figura 58 – Algoritmo *Que dia foi*.

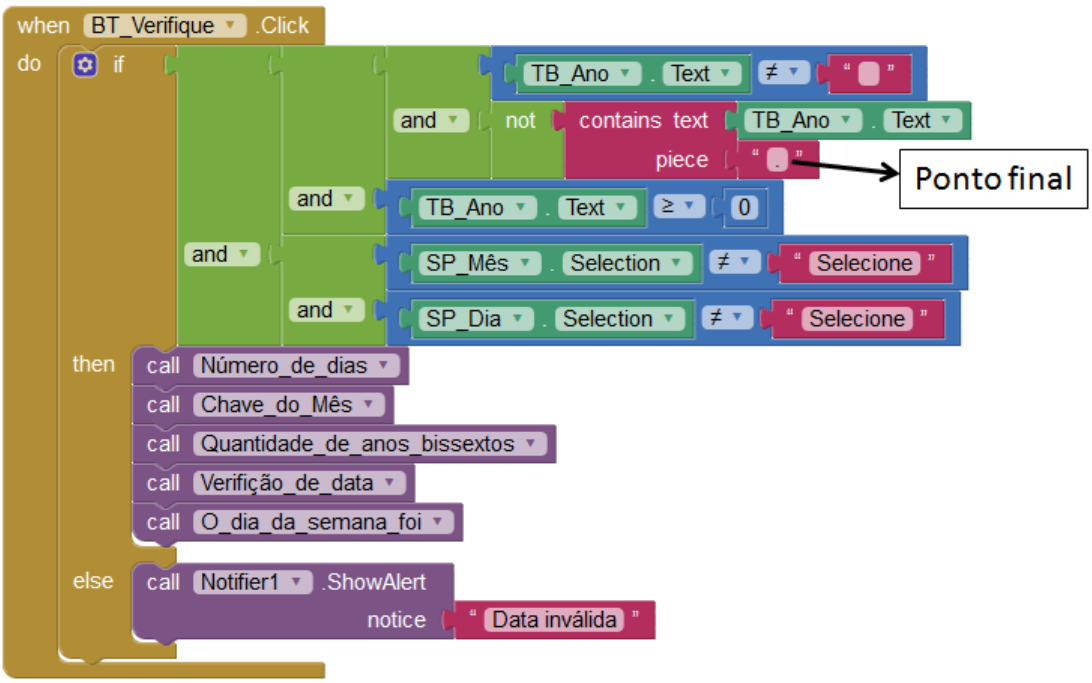


usuário digitar o ano e selecionar o mês e o dia. O algoritmo que configura as ações do botão "BT_Verifique" e previne possíveis erros de entrada de dados do usuário é ilustrado na Figura 59.

Apesar da caixa de texto ser configurada para inserir apenas números, há ainda possíveis erros que devem ser considerados, são eles: caixa de texto vazia, números negativos ou números decimais inseridos na caixa de texto. De modo a prevenir a entrada desses dados inválidos, o algoritmo verifica se a caixa de texto referente ao ano é diferente de uma *string* vazia, é verificado ainda se a caixa não contém um ponto final, para evitar números decimais e se o número contido na caixa de texto é maior do que ou igual a zero.

Para as seleções de mês e dia, basicamente, o único erro possível é que o usuário selecione uma opção inválida. Uma opção inválida seria o usuário não escolher um número, isto é, se a palavra Selecione fosse a opção escolhida, porém, para evitar que isso ocorra, é verificado se os

Figura 59 – Configuração do botão BT_Verifique.



Spinners referente ao mês e ao dia são diferentes de zero.

Caso essas verificações citadas sejam verdadeiras, o aplicativo deve executar os procedimentos criados até então, como apresenta a Figura 59.

Uma outra forma de opção inválida seria se o usuário selecionasse um dia inexistente a um determinado mês, por exemplo, o dia 31 de abril ou o dia 29 de fevereiro de um ano não bissexto são datas inexistentes. Apesar do procedimento referente a seleção do dia executar somente após o usuário definir o mês, é possível inserir um dia inexistente se o usuário selecionar por exemplo, o dia 31 de maio e posteriormente alterar a escolha do mês para abril. Para evitar essas situações basta que o aplicativo execute novamente o procedimento "Número_de_dias", através do bloco *call* "Número_de_dias", como ilustra a Figura 59.

Para finalizar, basta utilizar a ferramenta *Collapse Block* sobre o bloco *Click* do botão "BT_Verifique", finalizando a construção do aplicativo.

3 APLICATIVO CÁLCULO DE ÁREAS

Um dos principais assuntos abordados em geometria durante o ensino médio é o cálculo de áreas e são apresentadas aos estudantes diversas fórmulas para o cálculo das área de algumas figuras geométricas. Contudo o cálculo de área de polígonos não convexos, ou que não pertençam ao grupo dos triângulos e quadriláteros notáveis, não recebem a mesma atenção, embora sejam um tema mais próximo as necessidades do cotidiano. Neste capítulo, apresentamos o Teorema do Cadarço, também conhecido como Teorema de *Surveyor* ou Fórmula da Área de Gauss (SAMARAS, 2013, Acesso em: 20 out. 2017), como uma opção para o cálculo de áreas de polígonos e detalhamos o desenvolvimento de um aplicativo, intitulado Cálculo de Áreas. O desenvolvimento do aplicativo é uma alternativa de aprendizagem baseada em projetos, em que através da Teorema do Cadarço e das coordenadas dos vértices da região poligonal, obtidas pelo GPS do dispositivo móvel, calcula-se a área da região informada pelo usuário.

Este projeto envolve principalmente os conceitos matemáticos de operações vetoriais, determinantes de matrizes e cálculo de áreas, além das habilidades inerentes à programação.

3.1 MOTIVAÇÃO, DISCUSSÃO DO PROBLEMAS E OBJETIVOS

As formas geométricas estão presentes em nosso dia-a-dia, em especial, podemos destacar as formas geométricas da nossa casa, da quadra em que moramos, a forma geométrica de um campo de futebol, de uma escola, de uma cidade, entre outros. A área que todas essas formas geométricas ocupam podem ser calculadas através das fórmulas tradicionais, como as aplicadas por topógrafos.

A proposta deste capítulo é desenvolver um algoritmo que utiliza a latitude e longitude obtidas via GPS do aparelho celular, converte essas coordenadas para um sistema de coordenadas de um plano cartesiano bidimensional xy, através de uma projeção cilíndrica chamada UTM (Universal Transversa de Mercator). A partir dessas coordenadas, através de tópicos da geometria analítica, como vetores e determinantes de matrizes, calcula a área de uma região correspondente a um polígono simples utilizando o Teorema do Cadarço.

Espera-se que com essa atividade que o aluno aprofunde seus conhecimentos no cálculo de áreas, determinantes de matrizes e operações com vetores, perceba a matemática presente nos mais diversos aplicativos, além de desenvolver as habilidades inerentes à matemática e programação.

Acreditamos que a complexidade e a matemática envolvida na construção deste aplicativo é adequada a alunos do ensino médio. Além dos conteúdos matemáticos supracitados, outros conhecimentos matemáticos podem surgir ao longo do trabalho, bem como ideias para outros

aplicativos.

3.2 MATEMÁTICA DO APLICATIVO CÁLCULO DE ÁREAS

O cálculo de área a ser realizado pelo aplicativo só faz sentido se os polígonos forem simples, ou seja, os lados do polígono não podem se cruzar, assim os lados tem apenas os vértices como pontos em comum.

A sequência de resultados a seguir, apresentados em (CORRÊA, 2017) garante a possibilidade de triangularização dos polígonos simples.

Lema 3.1. *Todo polígono de n vértices com $n \geq 4$ possui uma diagonal.*

A demonstração pode ser encontrada em (CORRÊA, 2017).

Teorema 3.2. *Todo polígono simples P de n vértices pode ser particionado em triângulos pela adição de (zero ou mais) diagonais.*

Demonstração. A prova é feita por indução. Se $n = 3$, o polígono é um triângulo, e o teorema vale trivialmente. Seja $k \geq 3$, a hipótese de indução é que qualquer polígono simples com k vértices ou menos pode particionado em triângulos pela adição de diagonais. Vamos considerar um polígono P com $k + 1$ vértices. Como $k + 1 \geq 4$, existe $d = \overline{ab}$ uma diagonal do polígono P, segundo o Lema 3.1. A diagonal d divide o polígono P em dois polígonos P_1 e P_2 não vazios. Como o número de vértices de cada um dos polígonos P_1 e P_2 é menor que o número de vértices do polígono P, aplicamos a hipótese de indução nos dois polígonos e união das partições de P_1 e P_2 e da diagonal d, são uma partição em triângulos do polígono P, ou seja P admite uma triangularização. □

Lema 3.3. *Toda triangularização de um polígono P de n vértices consiste de $n - 3$ diagonais e $n - 2$ triângulos.*

Demonstração. Faremos a prova por indução. Para $n = 3$ as duas afirmações valem trivialmente. Seja $n \geq 4$, a hipótese de indução é que a triangularização de qualquer polígono simples com k vértices ou menos consiste de $k - 3$ diagonais e de $k - 2$ triângulos. Será provado que a triangularização de um polígono simples P de $k + 1$ vértices consiste de $(k + 1) - 3$ diagonais e $(k + 1) - 2$ triângulos.

Seja P um polígono com $k + 1$ vértices. Segundo o Lema 3.1, como $k + 1 \geq 4$, existe uma diagonal d que divide o polígono P em dois polígonos, P_1 e P_2. Sejam k_1 e k_2 o número de vértices dos polígonos P_1 e P_2, respectivamente. Temos que $k_1 + k_2 = (k + 1) + 2$ pois os vértices da diagonal d são contados duas vezes, pois pertencem a P_1 e P_2.

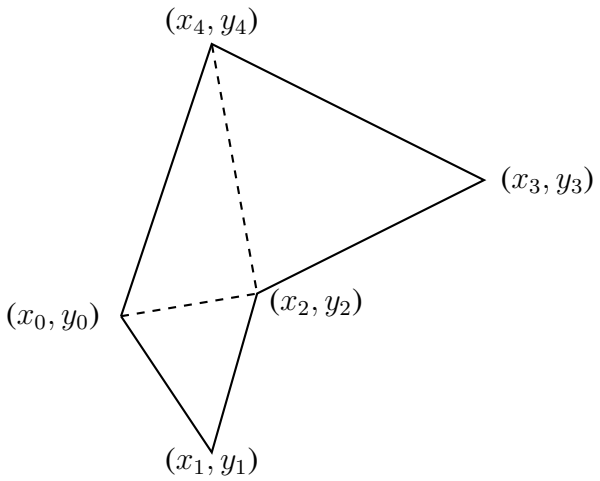
O número de diagonais D que fazem parte da triangularização do polígono P é igual a soma de diagonais dos polígonos que compõe a triangularização de P_1 e P_2 acrescidos da

diagonal d. Aplicando a hipótese de indução sobre o número de diagonais de P_1 e P_2, temos: $D = (k_1 - 3) + (k_2 - 3) + 1 = k_1 + k_2 - 5 = (k + 1) + 2 - 5 = (k + 1) - 3$ diagonais.

O número de triângulos T que fazem parte da triangularização do polígono P é igual a soma dos triângulos compõe a triangularização dos polígonos P_1 e P_2. Aplicando a hipótese de indução sobre o número de triângulos de P_1 e P_2, temos: $T = (k_1-2)+(k_2-2) = k_1+k_2-4 = (k + 1) + 2 - 4 = (k + 1) - 2$ triângulos. □

Um exemplo de triangularização pode ser observado na Figura 60. Neste polígono, as diagonais $(x_0, y_0)(x_2, y_2)$ e $(x_2, y_2)(x_4, y_4)$ fazem a triangularização.

Figura 60 – Exemplo de triangularização.

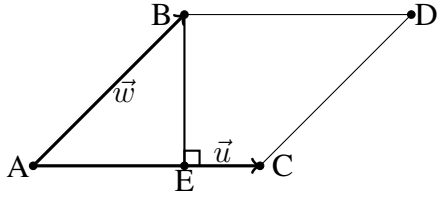


Garantida a possibilidade de triangularização de polígonos simples, vamos nos debruçar sobre o cálculo da área dos triângulos. Basearemo-nos nos resultados apresentados em (DELGADO, 2013).

Lema 3.4. *O valor absoluto do determinante* $\begin{vmatrix} \alpha & \beta \\ \alpha' & \beta' \end{vmatrix}$ *é a área do paralelogramo determinado pelos vetores* $\vec{u} = (\alpha, \beta)$ *e* $\vec{v} = (\alpha', \beta')$.

Demonstração. Seja o paralelogramo P da Figura 61. A área de P é obtida multiplicando a medida da base $|AC|$ pela altura $|EB|$. Se $\theta = \widehat{CAB}$, então $|EB| = |AB|sen\theta$, e portanto: Área de $P = |AB||AC|sen\theta$.

Figura 61 – Paralelogramo ABCD.



Utilizando linguagem vetorial e o produto interno, será obtida uma expressão para o cálculo da área do paralelogramo P. De fato, se $\vec{u} = \vec{AC}$ e $\vec{w} = \vec{AB}$, então:

$$\theta = \angle(\vec{u}, \vec{w}) \text{ e Área } P = ||\vec{u}||\ ||\vec{w}||sen\theta.$$

Sendo $sen^2\theta = 1 - cos^2\theta$, temos:

$$\begin{aligned}(\text{Área P})^2 &= (||\vec{u}|| \cdot ||\vec{w}|| \cdot sen\theta)^2 = ||\vec{u}||^2 \cdot ||\vec{w}||^2 \cdot sen^2\theta = ||\vec{u}||^2 \cdot ||\vec{w}||^2 \cdot (1 - cos^2\theta) \\ &= ||\vec{u}||^2 \cdot ||\vec{w}||^2 - ||\vec{u}||^2 \cdot ||\vec{w}||^2 \cdot cos^2\theta = ||\vec{u}||^2 \cdot ||\vec{w}||^2 - (||\vec{u}|| \cdot ||\vec{w}|| \cdot cos\theta)^2 \\ &= ||\vec{u}||^2 \cdot ||\vec{w}||^2 - \langle \vec{u}, \vec{w} \rangle.\end{aligned}$$

Portanto,

$$\text{Área P} = \sqrt{||\vec{u}||^2 \cdot ||\vec{w}||^2 - \langle \vec{u}, \vec{w} \rangle^2}.$$

Observa-se também que:

$$(\text{Área P})^2 = ||\vec{u}||^2 \cdot ||\vec{w}||^2 - \langle \vec{u}, \vec{w} \rangle^2 = det \begin{pmatrix} ||\vec{u}||^2 & \langle \vec{u}, \vec{w} \rangle \\ \langle \vec{u}, \vec{w} \rangle & ||\vec{w}||^2 \end{pmatrix} = det \begin{pmatrix} \langle \vec{u}, \vec{u} \rangle & \langle \vec{u}, \vec{w} \rangle \\ \langle \vec{u}, \vec{w} \rangle & \langle \vec{w}, \vec{w} \rangle \end{pmatrix}.$$

Se $\vec{u} = (\alpha, \beta)$ e $\vec{w} = (\alpha', \beta')$ em relação a um sistema de eixos ortogonais OXY, então:

$$||\vec{u}||^2 = \alpha^2 + \beta^2, ||\vec{w}||^2 = (\alpha')^2 + (\beta')^2 \text{ e } \langle \vec{u}, \vec{w} \rangle = \alpha \cdot \alpha' + \beta \cdot \beta',$$

e portanto,

$$\begin{aligned}(Area\ P)^2 &= (\alpha^2 + \beta^2)((\alpha')^2 + (\beta')^2) - (\alpha\alpha' + \beta\beta')^2 \\ &= \alpha^2(\alpha')^2 + \alpha^2(\beta')^2 + \beta^2(\alpha')^2 + \beta^2(\beta')^2 - \alpha^2(\alpha')^2 - 2\alpha\alpha'\beta\beta' - \beta^2(\beta')^2 \\ &= \alpha^2(\beta')^2 + \beta^2(\alpha')^2 - 2\alpha\alpha'\beta\beta' = (\alpha\beta')^2 - 2(\alpha\beta')(\beta\alpha') + (\beta\alpha')^2 \\ &= (\alpha\beta - \beta\alpha')^2 = \left[det \begin{pmatrix} \alpha & \beta \\ \alpha' & \beta' \end{pmatrix} \right]^2.\end{aligned}$$

Logo, a área do paralelogramo P, cujos lados adjacentes são representantes dos vetores $\vec{u} = (\alpha, \beta)$ e $\vec{w} = (\alpha', \beta')$, é igual ao módulo do determinante da matriz cujas linhas são as coordenadas de $\vec{u}$ e $\vec{w}$, respectivamente:

$$Area\ P = \left| det \begin{pmatrix} \alpha & \beta \\ \alpha' & \beta' \end{pmatrix} \right|.$$

Devido às propriedades de determinantes, a área também é igual ao módulo do determinante da matriz cujas colunas são as coordenadas de $\vec{u}$ e $\vec{w}$, respectivamente:

$$Area\ P = \left| det \begin{pmatrix} \alpha & \alpha' \\ \beta & \beta' \end{pmatrix} \right|.$$

□

Corolário 3.5. *Sejam ABC o triângulo de vértices A, B e C e $ABCD$ o paralelogramo de lados adjacentes AB e AC. Como os triângulos ABC e DCB são congruentes, temos:*

$$\textit{Área}(ABCD) = 2 \cdot \textit{Área}(ABC) = \left| det \begin{pmatrix} \overrightarrow{AB} \\ \overrightarrow{AC} \end{pmatrix} \right|,$$

onde $\begin{pmatrix} \overrightarrow{AB} \\ \overrightarrow{AC} \end{pmatrix}$ representa a matriz cujas colunas são as coordenadas dos vetores $\overrightarrow{AB}$ e $\overrightarrow{AC}$. Portanto,

$$\textit{Área}(ABC) = \frac{1}{2} \left| det \begin{pmatrix} \overrightarrow{AB} \\ \overrightarrow{AC} \end{pmatrix} \right|.$$

Ou ainda, se $A = (x_0, y_0)$,$B = (x_1, y_1)$ e $C = (x_2, y_2)$, então:

$$\textit{Área}(ABC) = \frac{1}{2} \begin{vmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{vmatrix}.$$

Nos fundamentaremos agora nos resultados anteriores, especialmente no Lema 3.2 e no Corolário 3.5 que mostram respectivamente, que é possível a triangularização de polígonos simples, o cálculo da área de polígonos para apresentar o resultado do Teorema do Cadarço, apresentado em (BRADEN; FRENSEL; CRISSAFF, 1986).

Teorema 3.6. *A área de um polígono simples de n lados, com os vértices (x_0, y_0), (x_1, y_1), ..., (x_{n-1}, y_{n-1}), listados em sentido anti-horário é:*

$$A = \frac{1}{2} \left\{ \begin{vmatrix} x_0 & x_1 \\ y_0 & y_1 \end{vmatrix} + \begin{vmatrix} x_1 & x_2 \\ y_2 & y_2 \end{vmatrix} + \cdots + \begin{vmatrix} x_{n-2} & x_{n-1} \\ y_{n-2} & y_{n-1} \end{vmatrix} + \begin{vmatrix} x_{n-1} & x_0 \\ y_{n-1} & y_0 \end{vmatrix} \right\}.$$

Demonstração. Do Corolário 3.5, sabe-se que a área A de um triângulo de vértices (x_0, y_0), (x_1, y_1) e (x_2, y_2) (no sentido anti-horário) é:

$$\begin{aligned}
A &= \frac{1}{2} \cdot \begin{vmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{vmatrix} \\
&= \frac{1}{2} \cdot [(x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)] \\
&= \frac{1}{2} \cdot [x_1y_2 - x_1y_0 - x_0y_2 + x_0y_0 - x_2y_1 + x_2y_0 + x_0y_1 - x_0y_0] \\
&= \frac{1}{2} \cdot \left[\underbrace{x_1y_2 - x_2y_1}_{A} \underbrace{-x_0y_2 + x_2y_0}_{B} \underbrace{+x_0y_1 - x_1y_0}_{C} \right] \\
&= \frac{1}{2} \cdot \left[\underbrace{\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix}}_{A} - \underbrace{\begin{vmatrix} x_0 & x_2 \\ y_0 & y_2 \end{vmatrix}}_{B} + \underbrace{\begin{vmatrix} x_0 & y_1 \\ y_0 & x_1 \end{vmatrix}}_{C} \right] \\
&= \frac{1}{2} \cdot \left[\begin{vmatrix} x_0 & y_1 \\ y_0 & x_1 \end{vmatrix} + \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & x_0 \\ y_2 & y_0 \end{vmatrix} \right].
\end{aligned}$$

Do Lema 3.3 todo polígono simples de n lados pode ser triangularizado em $n-2$ triângulos, utilizando $n-3$ diagonais auxiliares, onde cada diagonal é comum a dois triângulos adjacentes, mas orientadas em direções opostas em cada triângulo (pois os vértices de cada triângulo estão orientados no sentido anti-horário), então a área A orientada do polígono é a soma das áreas de cada triângulo.

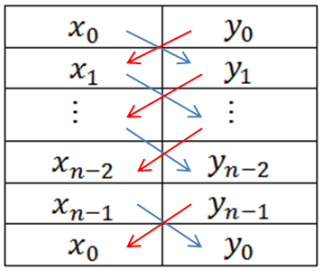
Aplicando então a fórmula da área dos triângulos a cada um dos triângulos, do polígono de n lados, com vértices no sentido anti-horário em $(x_0, y_0), (x_1, y_1), \cdots, (x_{n-2}, y_{n-2}),(x_{n-1}, y_{n-1})$, tem-se:

$$A = \frac{1}{2} \cdot \left(\begin{vmatrix} x_0 & x_1 \\ y_0 & y_1 \end{vmatrix} + \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} + \cdots + \begin{vmatrix} x_{n-2} & x_{n-1} \\ y_{n-2} & y_{n-1} \end{vmatrix} + \begin{vmatrix} x_{n-1} & x_0 \\ y_{n-1} & y_0 \end{vmatrix} \right). \tag{3.1}$$

□

Organizando as coordenadas em um quadro, é possível observar na Figura 62 a semelhança com um cadarço, onde as setas azuis indicam os produtos das diagonais principais das matrizes e as setas vermelhas indicam os produtos das diagonais secundárias.

Figura 62 – Teorema do Cadarço.



É possível notar na Equação 3.1 a ausência dos determinantes correspondentes às $n-2$ diagonais, isso se deve ao fato de que, como cada diagonal tem orientação oposta em cada um dos dois triângulos a que é adjacente, seus determinantes anulam-se.

Por exemplo, suponha a diagonal (x_i, y_i) (x_j, y_j), em um dos triângulos adjacentes o determinante é $\begin{vmatrix} x_i & x_j \\ y_i & y_j \end{vmatrix}$ e no outro triângulo, o determinante é $\begin{vmatrix} x_j & x_i \\ y_j & y_i \end{vmatrix}$ como $\begin{vmatrix} x_i & x_j \\ y_i & y_j \end{vmatrix} + \begin{vmatrix} x_j & x_i \\ y_j & y_i \end{vmatrix} = 0$, as matrizes correspondentes às diagonais anulam-se.

3.3 SISTEMA UTM

O Sistema de Projeção Universal Transversa de Mercator, ou sistema UTM, utilizado desde 1950 é uma projeção cilíndrica e recebe este nome pois é adotada internacionalmente para representação da superfície da Terra, é transversa porque os cilindros que envolvem a Terra na projeção são transversais ao eixo rotacional da Terra e Mercator porque foi Gerardus Mercator

que introduziu os conceitos de projeções cilíndricas. O sistema de UTM tem como objetivo minimizar as distorções causadas pela projeção do elipsóide que representa o planeta Terra em um sistema ortogonal a níveis aceitáveis. (SZNELWAR, 2007, Acesso em: 16 ago. 2017)

A projeção cilíndrica envolve diversas fórmulas que não são pertinentes aos assuntos abordados nos ensinos fundamental e médio e além disso, envolvem conceitos de cartografia que não são objetos deste trabalho e por essas razões, as deduções das fórmulas não serão apresentadas neste trabalho.

Basicamente temos coordenadas de latitude e longitude, utilizaremos os parâmetros de um Datum Geodésico, que basicamente consiste em adotar dados de um elipsóide de referência que representa uma região. O Datum adotado foi o *WGS84*, idêntico ao Datum Sirgas 2000, sistema de referência adotado nas Américas do Sul, Central e Norte (MARINO, 2013, Acesso em: 10 nov. 2017). O Datum *WGS84* é o sistema de referência utilizado pelos GPS's e, através dele, obtemos coordenadas planas bidimensionais x e y, através das fórmulas em (MADEIRA, 2013, Acesso em: 16 ago. 2017), descritas a seguir:

$$x = k_0 \cdot N \cdot \left(A + \frac{(1 - T + C) \cdot A^3}{6} + \frac{(5 - 18 \cdot T + T^2 + 72 \cdot C - 58 \cdot e'^2) \cdot A^5}{120} \right) + FE.$$

$$y = k_0 \cdot \left(M - M_0 + N \cdot Tan(\phi) \cdot \left(\frac{A^2}{2} + \frac{(5 - T + 9 \cdot C + 4 \cdot C^2) \cdot A^4}{24} + \frac{(61 - 58 \cdot T + T^2 + 600 \cdot C - 330 \cdot e'^2) \cdot A^6}{720} \right) \right) + FN.$$

Onde:

$$N = \frac{a}{\sqrt{1 - e^2 \cdot Sen^2(\phi)}}. \tag{3.2}$$

$$T = Tan^2(\phi). \tag{3.3}$$

$$C = e'^2 \cdot Cos^2(\phi). \tag{3.4}$$

$$A = (long - zcm) \cdot \frac{\pi}{180} \cdot Cos(\phi). \tag{3.5}$$

$$M = (M_1 - M_2 + M_3 - M_4) \cdot a \tag{3.6}$$

$$M_1 = \phi \cdot \left(1 - \frac{e^2}{4} - \frac{3 \cdot e^4}{64} - \frac{5 \cdot e^6}{256} \right) \tag{3.7}$$

$$M_2 = Sen(2 \cdot \phi) \cdot \left(\frac{3 \cdot e^2}{8} + \frac{3 \cdot e^4}{32} + \frac{45 \cdot e^6}{1024} \right) \tag{3.8}$$

$$M_3 = Sen(4 \cdot \phi) \cdot \left(\frac{15 \cdot e^4}{256} + \frac{45 \cdot e^6}{1024} \right) \tag{3.9}$$

$$M_4 = Sen(6 \cdot \phi) \cdot \left(\frac{35 \cdot e^6}{3072} \right) \tag{3.10}$$

$$k_0 = 1 - \frac{1}{2500} = 0,996$$

$$utmz = 1 + \frac{long + 180}{6} \tag{3.11}$$

$$zcm = 3 + 6 \cdot (utmz - 1) - 180 \tag{3.12}$$

$$\phi = lat \cdot \frac{\pi}{180} \tag{3.13}$$

$$\lambda = long \cdot \frac{\pi}{180} \tag{3.14}$$

Observação: Os valores de *lat* e *long* são respectivamente os valores de latitude e longitude em graus decimais e os valores de ϕ e λ são respectivamente os valores de latitude e longitude em radianos.

Os parâmetros do elipsoide de referência *WGS84* são os seguintes:

Raio equatorial em metros: $a = 6378137,0$

Raio polar em metros: $b = 6356752,3142$

Achatamento polar: $f = \frac{a-b}{a} = 0.0033528106647474805$

Primeira excentricidade: $e = \frac{\sqrt{a^2-b^2}}{a} = 0,081819190842622$

Primeira excentricidade ao quadrado: $e^2 = 0,00669437999014$

Segunda excentricidade: $e' = \frac{\sqrt{a^2-b^2}}{b} = 0,082094437949696$

Segunda excentricidade ao quadrado: $e'^2 = \frac{a^2-b^2}{b^2} = 0,00673949674228$

3.4 IMPLEMENTAÇÃO DO APLICATIVO CÁLCULO DE ÁREAS

É preciso calcular a área de uma região correspondente a um polígono simples, definido pelas coordenadas do GPS, porém, o GPS fornece as coordenadas em termos de latitude e

longitude. A estratégia adotada para solucionar esse problema foi utilizar a projeção cilíndrica, na qual o elipsoide correspondente à Terra é projetado sobre 60 cilindros (um cilindro para cada fuso de 6 graus) e as coordenadas são obtidas em termos de x e y, em um plano cartesiano.

Uma vez obtidas as coordenadas é necessário determinar a área do polígono formado por esses vértices. A estratégia adotada para determinar a área foi utilizar o Teorema do Cadarço que consiste basicamente em operações de adição, subtração e multiplicação das coordenadas obtidas.

O método utilizado no processo do cálculo de área deste aplicativo está sujeito a erros provenientes de diversos fatores, pois, o GPS tem atualmente precisão com raio de 3 a 5 metros (KASTRENAKES, 2017, Acesso em: 04 nov. 2017), a distorção da projeção cilíndrica é mais acentuada quando nos aproximamos dos polos (VASCONCELLOS, 2017, Acesso em: 20 nov. 2017) e o erro proveniente do arredondamento característico do uso de recursos computacionais.

Estes erros são inerentes às tecnologias digitais e são situações que não precisam ser evitadas, pelo contrário, é possível explorar essas limitações técnicas paralelamente ao desenvolvimento dos aplicativos, para que esses erros não sejam obstáculos durante o processo de aprendizagem, mas sim uma oportunidade para o aprofundamento na reflexão e motivação para buscar justificativas. (GIRALDO; CAETANO; MATTOS, 2013)

As fórmulas apresentadas na Seção 3.3 serão implementadas nos algoritmos através da sequência de algoritmos construídos no *MIT App Inventor 2*, a seguir.

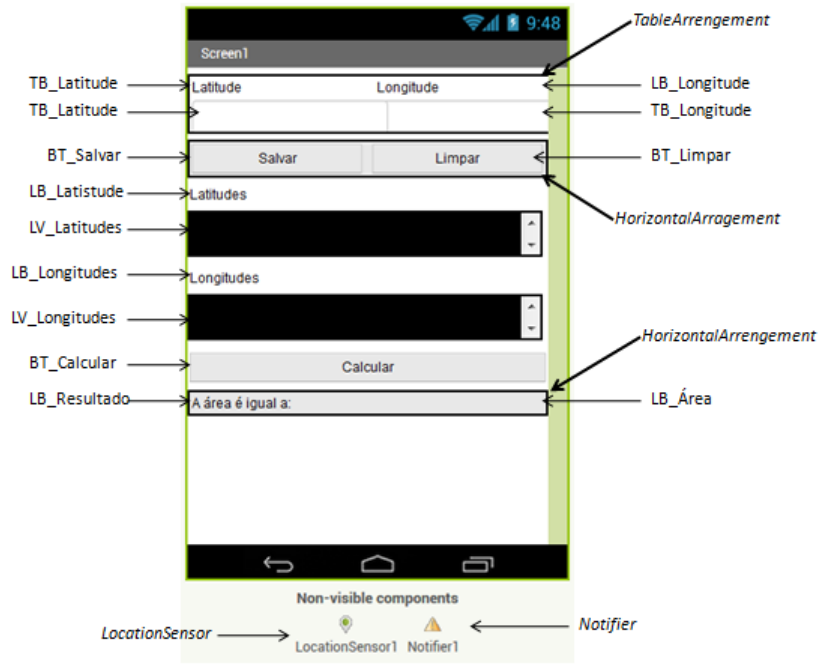
3.4.1 CONFIGURAÇÃO DA TELA DE DESIGNER

Em nossa implementação, os blocos da tela de *Designer* foram posicionados como apresentado na Figura 63.

Para a tela de *Designer* fiquem como na Figura 63, serão necessários os seguintes blocos e configurações:

- **Botões** : três botões, renomeando-os como “BT_Salvar”, “BT_Limpar” e “BT_Calcular”, alterar a propriedade *Text* para Salvar, Limpar e Calcular, respectivamente. Alteara a propriedade *Width* de todos eles para *Fill parent* e desmarcar a propriedade *Enable* do botão BT_Calcular;
- ***Labels***: seis *labels*, renomeando-os como “LB_Latitude”, “LB_Longitude”, “LB_Latitudes”, “LB_Longitudes”, “LB_Resultado” e “LB_Área”. Alterar a propriedade *Text* para Latitude, Longitude, Latitudes, Longitudes, A área é igual a:, respectivamente e deixar a propriedade *Text* do *label* “LB_Área” em branco;
- ***ListView***: dois *ListView*, renomeando-os para “LV_Latitudes” e “LV_Longitudes”;
- ***Notifier***: um bloco *Notifier*;

Figura 63 – Tela de *Designer* do aplicativo.

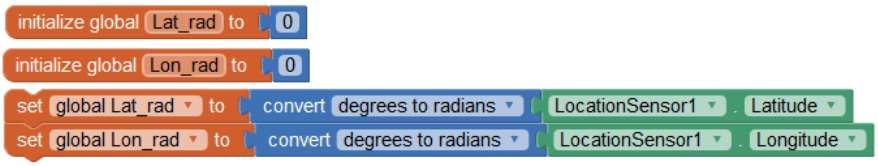


- **Caixas de texto**: duas caixas de texto, renomeando-as para “TB_Latitude” e “TB_Longitude”;
- ***HorizontalArrangement***: dois blocos de arranjo horizontal, alterando suas propriedades *Width* para *FillParent*;
- ***TableArrangement***: um bloco *TableArrangement*;
- ***LocationSensor***: um bloco *LocationSensor*;

3.4.2 LONGITUDE, LATITUDE EM RADIANOS

Será necessário converter para radianos os valores de latitude e longitude que são fornecidos em graus decimais para radianos, foram criadas duas variáveis globais, chamadas “Lat_rad” e “Long_rad” para armazenar os valores convertidos. O algoritmo que realiza essa conversão é apresentado na Figura 64.

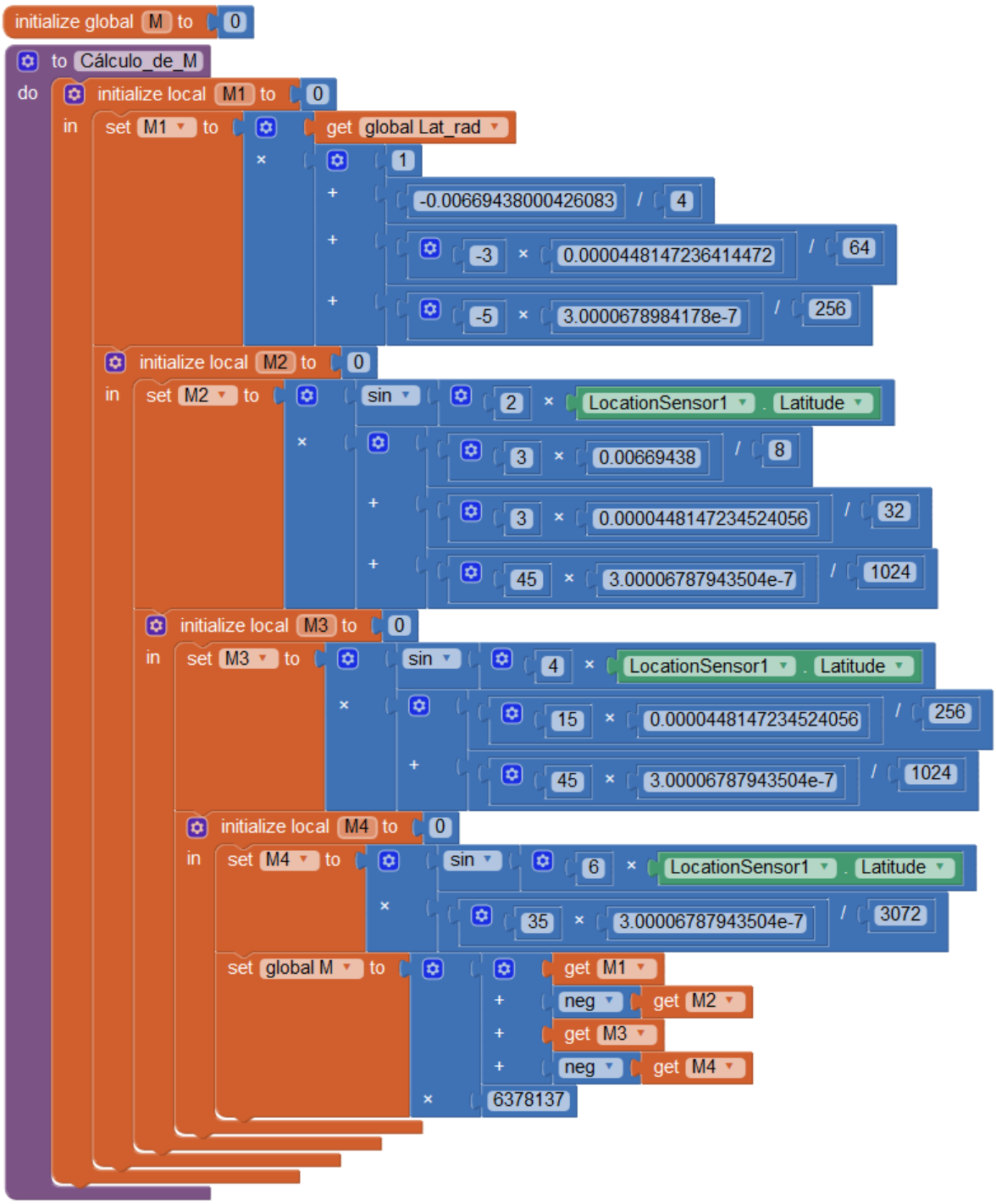
Figura 64 – Conversão de latitude e longitude em radianos.



3.4.3 CÁLCULO DE M

Para calcular o valor de M e as fórmulas que determinam os respectivos valores da série, M_1, M_2, M_3 e M_4, foi criado o algoritmo da Figura 65.

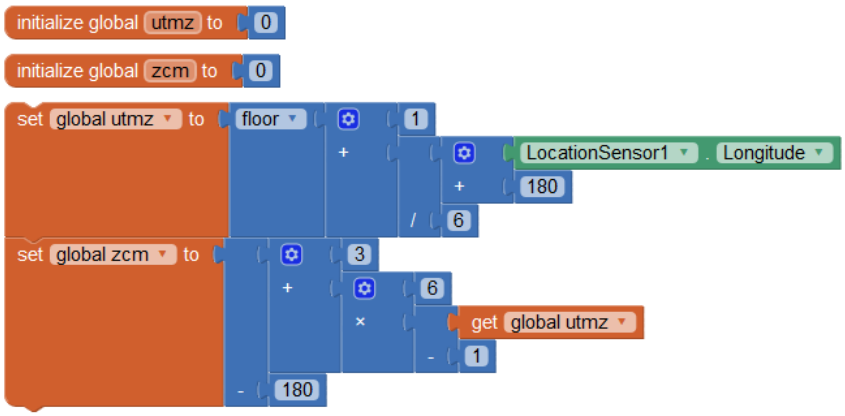
Figura 65 – Algoritmo para o cálculo de M.



3.4.4 ZONA UTM E MERIDIANO CENTRAL

Para calcular a Zona UTM, representada pela sigla utmz e seu respectivo meridiano central, representado pela sigla zcm, foi criado o algoritmo da Figura 66.

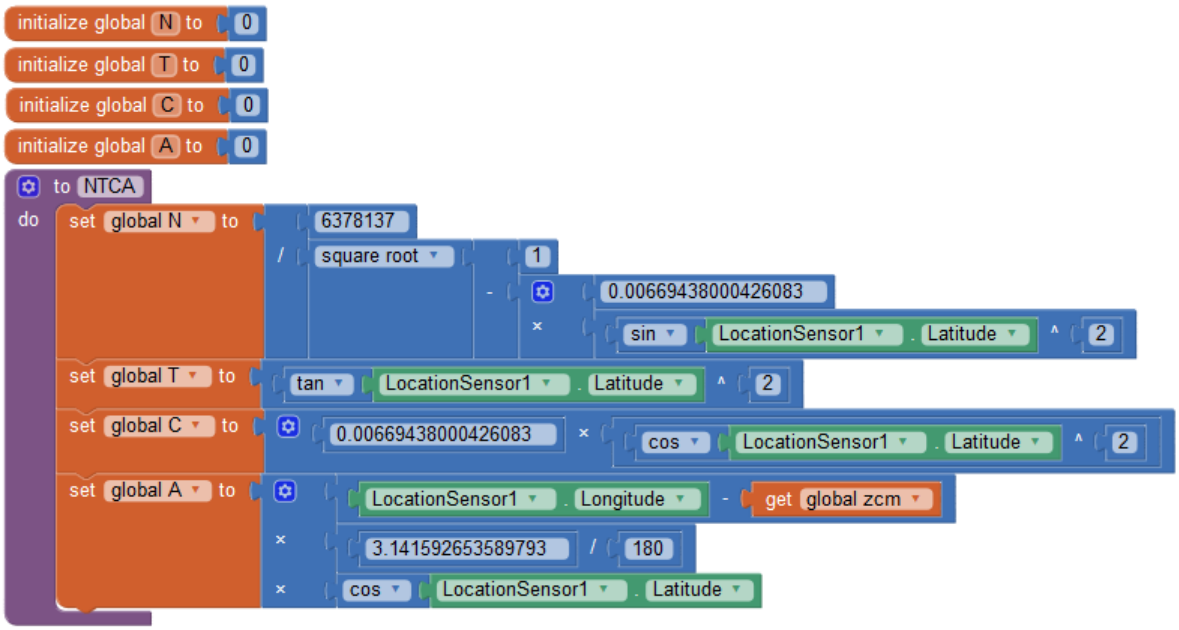
Figura 66 – Zona UTM e meridiano central.



3.4.5 CÁLCULO DE NTCA

Para implementar as fórmulas das variáveis N, T, C, A, foi criado o algoritmo da Figura 67.

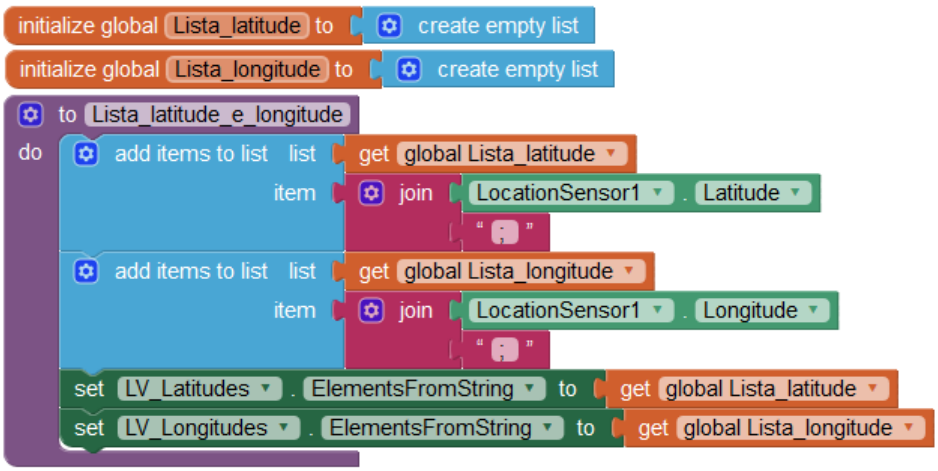
Figura 67 – Cálculo de $NTCA$.



3.4.6 LISTA DE LATITUDE E LONGITUDE

Para que seja possível o usuário verificar quais pontos estão sendo considerados para o cálculo da área, as latitudes e longitudes serão listadas nos *ListView's* "LV_Latitude" e "LV_Longitude". Para isso, foi criado o algoritmo da Figura 68.

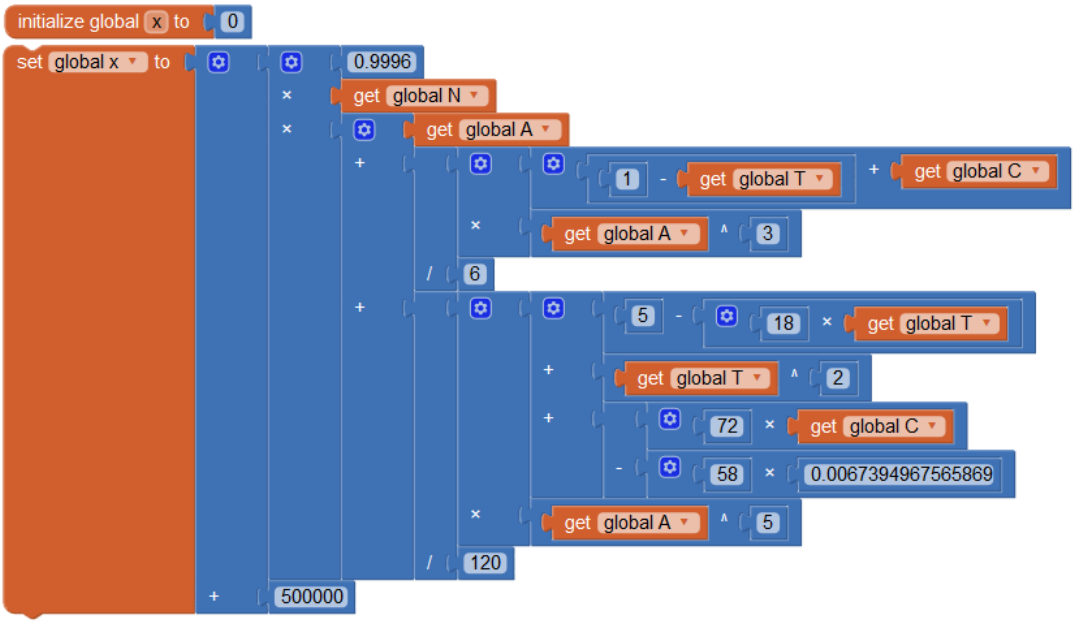
Figura 68 – Criação das listas de latitude e longitude.



3.4.7 CÁLCULO DE X

Para implementar a fórmula correspondente ao cálculo da coordenada x, foi desenvolvido o algoritmo da Figura 69.

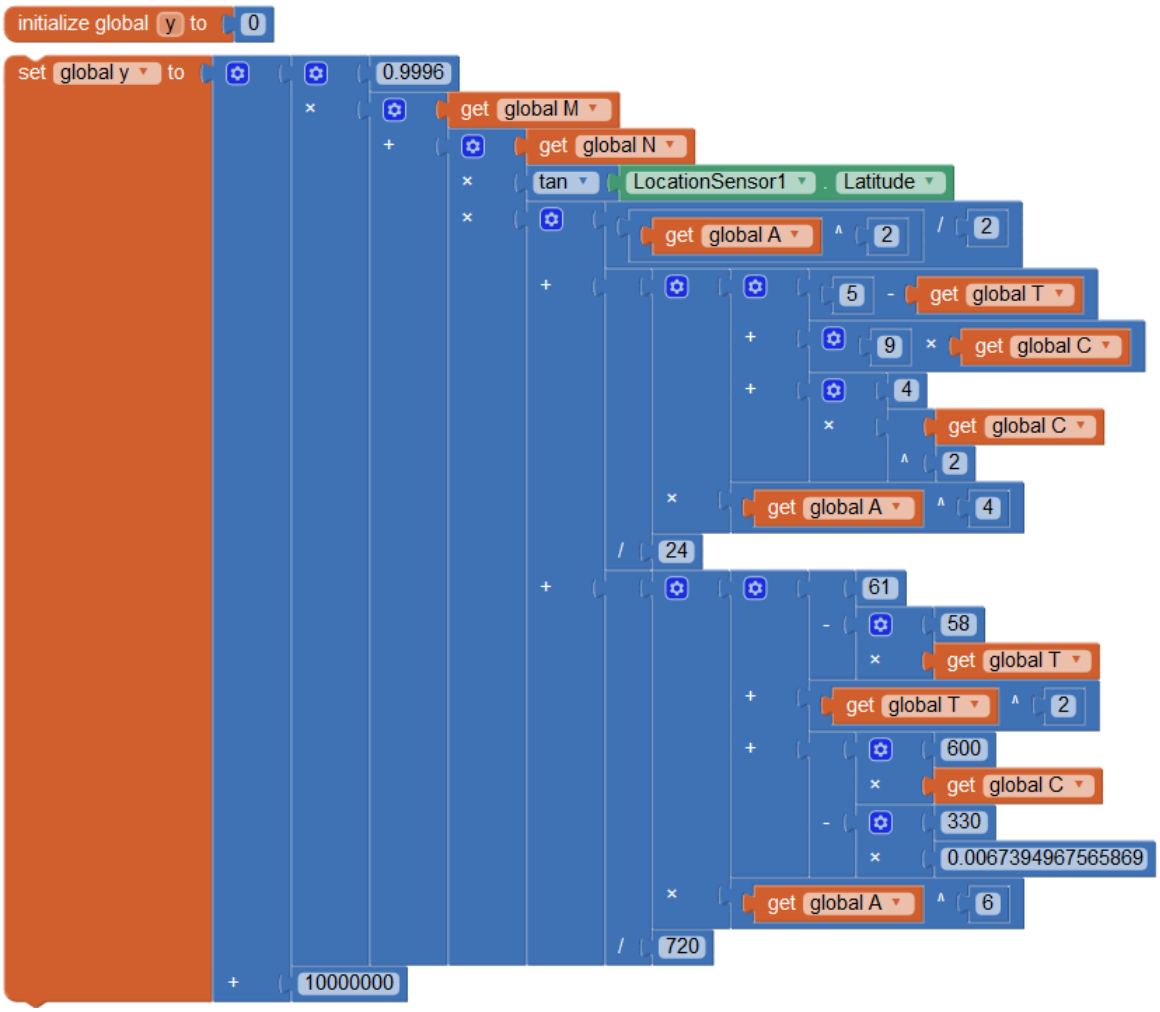
Figura 69 – Cálculo da coordenada x.



3.4.8 CÁLCULO DE Y

Para implementar a fórmula que determina a coordenada y, foi desenvolvido o algoritmo da Figura 70.

Figura 70 – Cálculo da coordenada y.



3.4.9 LISTA DAS COORDENADAS E PROCEDIMENTO

Para realizar o cálculo da área através do teorema do cadarço, é necessário criar uma lista para cada uma das coordenadas. Para isso foi criado o algoritmo da Figura 71.

Figura 71 – Lista das coordenadas x e y.

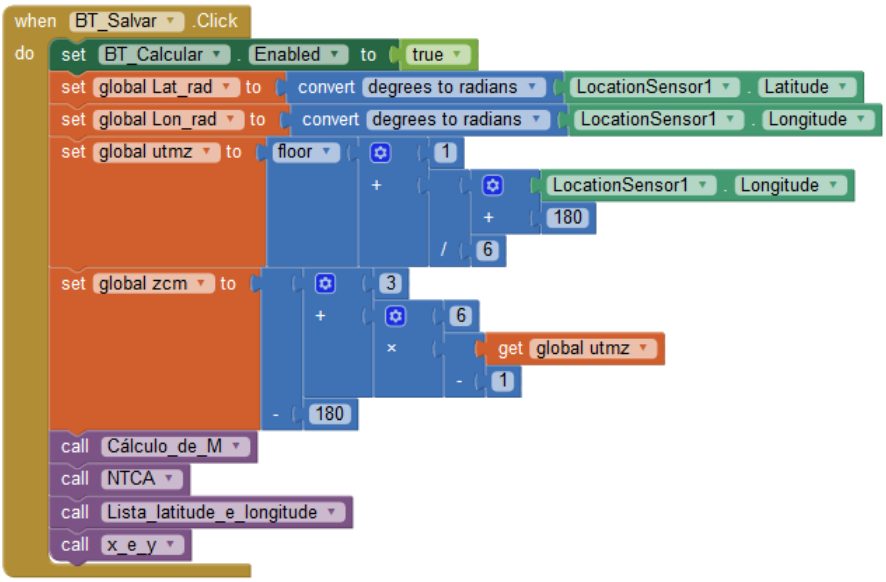
Observe que foi utilizada a ferramenta *CollapseBlock* nos blocos correspondentes aos

algoritmos dos cálculos de x e y.

3.4.10 BOTÃO SALVAR

Quando o botão salvar for clicado, a latitude e longitude devem ser convertidas para radianos, calcular os valores de utmz e zcm, chamar o procedimento correspondente ao cálculo de M e seus parâmetros, o procedimento correspondente ao cálculo das variáveis N, T, C e A, o procedimento que lista os valores latitude e longitude e o procedimento correspondente ao cálculo de x e y e os lista. Para realizar esses procedimentos, foi criado o algoritmo da Figura 72.

Figura 72 – Botão salvar.



O botão "BT_Calcular" foi criado na Seção 3.4.1 inicialmente desabilitado. Isso foi necessário para evitar que o usuário clique o botão "BT_Calcular" sem ao menos ter selecionado um par de coordenadas de latitude e longitude.

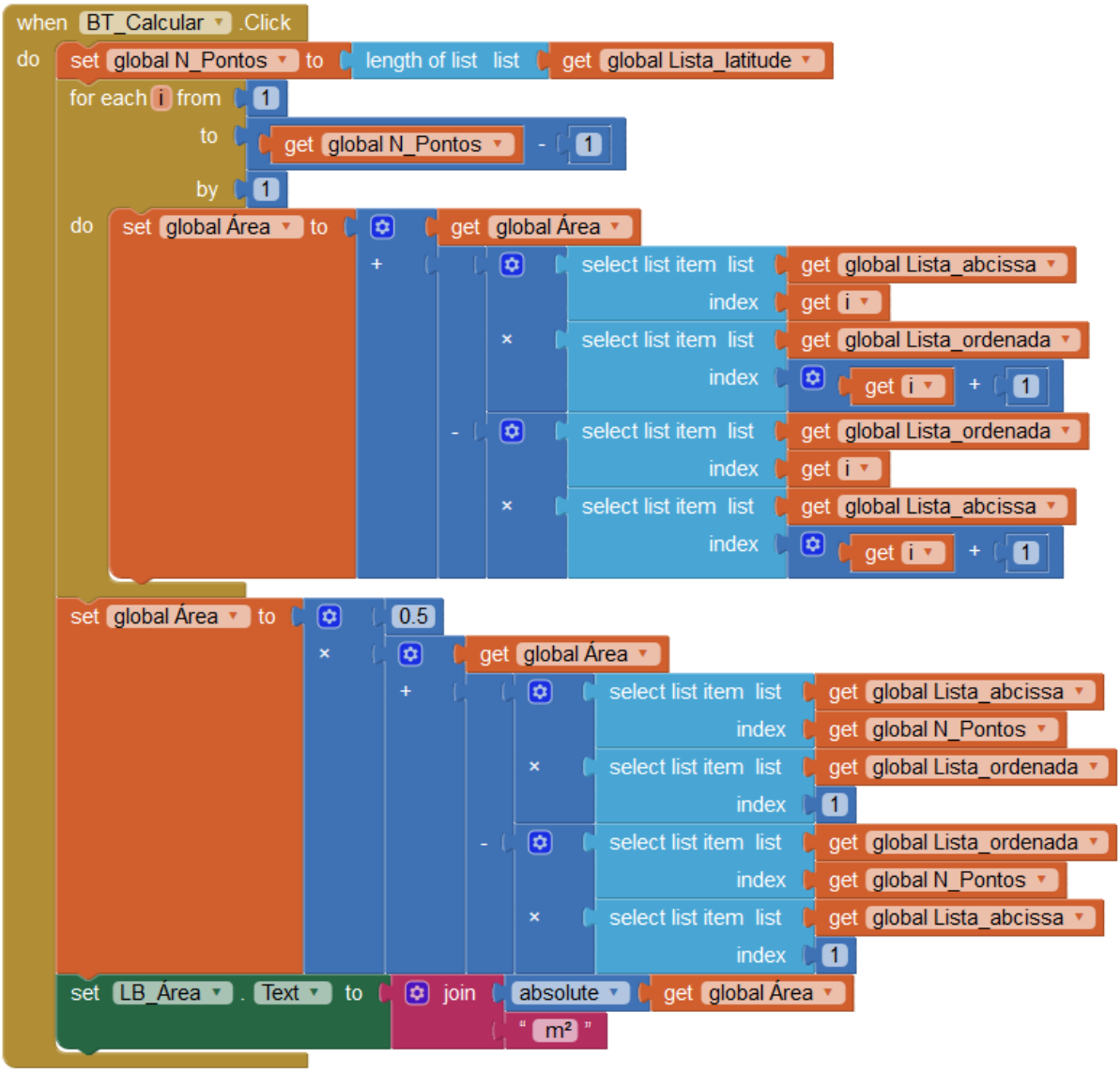
3.4.11 BOTÃO CALCULAR E O TEOREMA DO CADARÇO

Quando o usuário clicar o botão calcular, o Teorema do Cadarço deve ser executado. O algoritmo que o faz é apresentado na Figura 73.

Esse algoritmo faz o seguinte: para cada valor de "i", que vai de 1 até o número de pontos selecionados subtraído de uma unidade, a variável Área recebe o valor da própria variável Área somada à subtração do produto do elemento de ordem i da lista de abscissas pelo elemento de ordem $i + 1$ da lista de ordenadas pelo produto do elemento de ordem i da lista de ordenada pelo elemento de ordem $i + 1$ da lista de abcissas.

Após encerrar este laço de repetição, o produto do último elemento da lista de abscissas pelo primeiro elemento da lista de ordenadas subtraído do produto do último elemento da lista

Figura 73 – Teorema do cadarço.

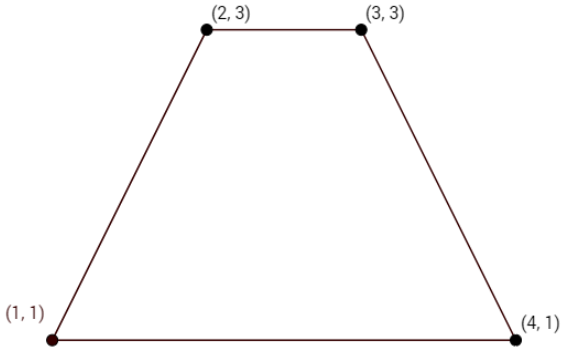


de ordenadas pelo primeiro elemento da lista de abscissas é somado ao valor da variável Área, obtido do laço de repetição e o produto dessa soma com 0.5 são o resultado do cálculo da área.

Por exemplo, suponha que foram selecionados 4 pontos, então a variável "i" vai de 1 até 3 (pois o laço for está definido para ir até N_Points $-$ 1). Suponha também que os pares ordenados (x, y) são $(1, 1)$, $(2, 3)$, $(3, 3)$ e $(4, 1)$, vértices do trapézio da Figura 74. Temos que os elementos da lista de abcissas são: 1,2,3 e 4 e os elementos da lista de ordenadas são 1,3,3 e 1. Tem-se então que:

- **Para** $i = 1$: A variável Área recebe o valor da própria variável Área que é igual a 0, e soma à seguinte expressão: $1 \cdot 3 - 1 \cdot 2 = 1$, logo a variável Área tem atualmente, valor igual a 1.
- **Para** $i = 2$: A variável Área recebe o valor da própria variável Área que é igual a 1, e

Figura 74 – Trapézio de vértices (1,1), (2,3), (3,3) e (4,1).



soma à seguinte expressão: $2 \cdot 3 - 3 \cdot 3 = -3$, logo a variável Área tem atualmente, valor igual a -2.

- **Para** $i = 3$: A variável Área recebe o valor da própria variável Área que é igual a -2, e soma à seguinte expressão: $3 \cdot 1 - 3 \cdot 4 = -9$, logo a variável Área tem atualmente, valor igual a -11, encerrando o laço *for*.

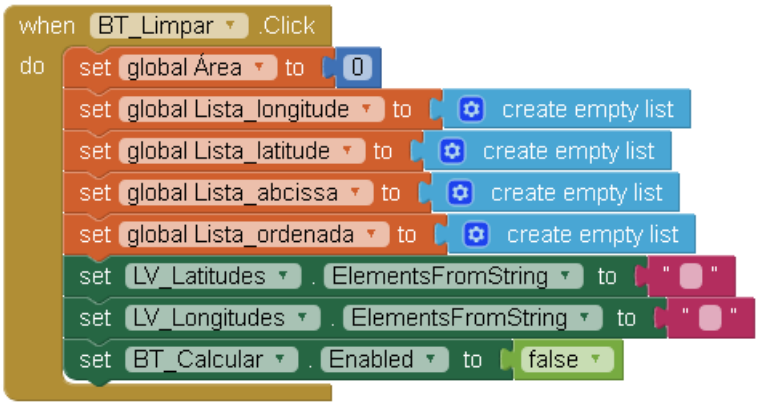
Após encerrado o laço for, a variável Área que tem valor igual a -11 é somada à seguinte expressão: $4 \cdot 1 - 1 \cdot 1 = 3$ e essa soma é multiplicada por 0.5. Desse modo, o novo valor da variável Área corresponde ao resultado da seguinte expressão: $0.5 \cdot (-11 + 3) = -4$.

O bloco *absolute* retorna o valor positivo da entrada a ele conectada, por isso, o *label* Área recebe valor igual a 4m².

3.4.12 BOTÃO LIMPAR E ALTERAÇÕES DE LATITUDE E LONGITUDE

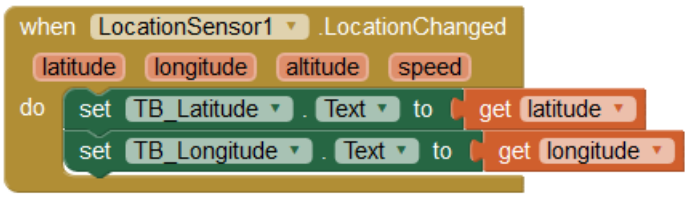
Para limpar os dados salvos no aplicativo e realizar uma nova coleta de latitudes e longitudes foi criado o algoritmo da Figura 75.

Figura 75 – Botão Limpar.



Quando o sensor *LocationSensor1* alterar sua posição, é interessante informar o usuário que isso aconteceu. Para isso, foi criado o algoritmo da Figura 76.

Figura 76 – Alteração de posição do sensor.



4 CONCLUSÕES

Neste trabalho apresentamos um breve tutorial de introdução ao *MIT App Inventor 2*, uma plataforma *on-line* e gratuita para o desenvolvimento de aplicativos para dispositivos Android que utiliza como forma de programação o encaixe de blocos para a construção dos algoritmos.

A proposta deste trabalho foi apresentar dois aplicativos onde os próprios alunos do ensino fundamental e médio pudessem desenvolvê-los com o auxílio do professor. Os aplicativos foram pensados de modo que a Matemática seja fundamental para os aplicativos, porém esteja implícita nos algoritmos, ou seja, os aplicativos propostos não limitam-se a resolver problemas matemáticos do cotidiano da sala de aula, mas solucionam situações mais próximas de questões reais do dia-a-dia dos alunos.

Propusemos inicialmente a construção do aplicativo *Que Dia Foi*, voltado principalmente para alunos do ensino fundamental, onde utilizando conceitos matemáticos de divisibilidade e congruência, os alunos elaborem uma série de algoritmos que, operando em conjunto determinam o dia da semana correspondente a uma data desejada.

O segundo aplicativo proposto é voltado principalmente para os alunos do ensino médio. O aplicativo recebeu o nome *Cálculo de Áreas* e utiliza o recurso do GPS do aparelho celular para coletar coordenadas de latitude e longitude em graus e após realizar a conversão para um plano cartesiano xy, utiliza o Teorema do Cadarço para determinar a área da região correspondente à área determinada pelas coordenadas coletadas.

A necessidade de saber o dia da semana correspondente a uma data é uma questão comum do nosso cotidiano, pois recorrentemente é interessante saber o dia da semana de uma data específica, como eventos, feriados e datas comemorativas. As áreas de polígonos simples também fazem parte do nosso dia-a-dia pois estamos cercados de formas geométricas e a utilização do Teorema do Cadarço nos permite determinar a área de figuras geométricas que normalmente não são estudadas em sala de aula, figuras essas correspondentes ao formato do terreno de nossas casas, nossos bairros e escolas.

Já existem diversos aplicativos, inclusive gratuitos que executam as mesmas funções dos aplicativos propostos neste trabalho, entretanto quando o próprio aluno cria seus próprios aplicativos, além de despertar o interesse dos alunos na lógica de programação e desenvolver as habilidades inerentes à matemática e programação, o aluno é capaz de perceber que a matemática está presente nos mais diversos aplicativos e reconhecer sua importância nas atividades cotidianas.

Os aplicativos implementados podem ser baixados através dos seguintes endereços:

- Aplicativo *Que Dia Foi*:
  <ai2.appinventor.mit.edu/?galleryId=6021161103917056>

- Aplicativo *Cálculo de Áreas*:
  <ai2.appinventor.mit.edu/?galleryId=6602617266110464>

REFERÊNCIAS

BRADEN, B.; FRENSEL, K.; CRISSAFF, L. **The Surveyor's area formula**. Disponível em:http://steiner.math.nthu.edu.tw/disk5/js/cardioid/12.pdf, 1986.

CALENDARIO gregoriano. Disponível em:http://www.calendariodoano.com.br/calendario-gregoriano/, 2017, Acesso em: 21 jun. 2017.

CORRÊA, L. **Teorema da galeria de arte e triangularização de polígonos e pontos no plano**. Disponível em:https://www.ime.usp.br/ cpq/main/arquivos/teoremadagaleriadearteetriangularizacaodepoligonos.pdf, 2017.

DELGADO, J. **Geometria analitica**. Rio de Janeiro: Sociedade Brasileira de Matemática, 2013.

FONSECA, A.; ALMADA, T. **Números inteiros: alguns critérios de divisibilidade**. Disponível em:http://gazeta.spm.pt/getArtigo?gid=419, 2013.

GIRALDO, V.; CAETANO, P.; MATTOS, F. **Recursos computacionais no ensino de matemática**. Rio de Janeiro: Sociedade Brasileira de Matemática, 2013.

HARDESTY, L. **The MIT roots of Google's new software**. Disponível em:http://news.mit.edu/2010/android-abelson-0819, 2010, Acesso em: 05 fev. 2017.

HEFEZ, A. **Aritmetica**. Rio de Janeiro: Sociedade Brasileira de Matemática, 2014.

HIGA, P. **95.5% dos smartphones vendidos no Brasil são Androids**. Disponível em: https://tecnoblog.net/203749/android-ios-market-share-brasil-3t-2016/, 2016, Acesso em: 10 dez. 2016.

KAREN. **Computational thinking concepts with App Inventor for primary school children**. Disponível em: http://appinventor.mit.edu/explore/blogs/karen/2016/09-2.html, 2016, Acesso em: 05 fev. 2017.

KASTRENAKES, J. **GPS will be accurate within one foot in some phones next year**. Disponível em:https://www.theverge.com/circuitbreaker/2017/9/25/16362296/gps-accuracy-improving-one-foot-broadcom, 2017, Acesso em: 04 nov. 2017.

MADEIRA, D. **Conversão entre coordenadas geográficas e UTM**. Disponível em:http://danscientia.blogspot.com.br/2013/05/conversao-entre-coordenadas-geograficas.html, 2013, Acesso em: 16 ago. 2017.

MARINO, T. **Conceitos de geodésica**. Disponível em:http://www.ufrrj.br/lga/tiagomarino/aulas/32013, Acesso em: 10 nov. 2017.

MICROSFOT CORPORATION. **Dia da semana função dia da semana**. Disponível em:https://support.office.com/pt-br/article/DIA-DA-SEMANA-Fun2017, Acesso em: 20 jun. 2017.

SAMARAS, C. **Calculating the area of a simple polygon using the shoelace algorithm**. Disponível em:http://www.myengineeringworld.net/2014/06/polygon-area-shoelace-algorithm-Excel.html, 2013, Acesso em: 20 out. 2017.

SETTING up App Inventor. Disponível em:http://appinventor.mit.edu/explore/ai2/setup.html, 2012, Acesso em: 10 fev. 2017.

SOUSA, D. **Como descobrir o dia da semana em que você nasceu**. Disponível em:http://gigamatematica.blogspot.com.br/2013/03/como-descobrir-o-dia-da-semana-em-que.html, 2013, Acesso em: 20 jun. 2017.

SZNELWAR, M. **Sistema UTM**. Disponível em:http://www.leg.ufpr.br/lib/exe/fetch.php/disciplinas/verao2007/pdf/sistema_utm.pdf, 2007, Acesso em: 16 ago. 2017.

VASCONCELLOS, M. **Projeção de Mercator**. Disponível em:https://www.infoescola.com/cartografia/projecao-de-mercator/, 2017, Acesso em: 20 nov. 2017.