

Desenvolvimento de Componentes de Visão Computacional para Programação por Blocos

ImageCV - Componente de Visão Computacional para o App Inventor

Introdução

Visão computacional está se tornando uma área cada vez mais utilizada na sociedade atual. Aplica-se a reconhecimento de sorriso em câmeras fotográficas, reconhecimento facial em sistemas de vigilância, reconhecimento de culturas de plantas em fotografias aéreas e outras.

Trabalhar nessa área requer conhecimento em computação, Inteligência Artificial, processamento de imagens e na área de aplicação. A conjunção das habilidades para o trabalho geralmente é conseguida formando equipes multidisciplinares e, mais recentemente, dando formação interdisciplinar aos profissionais.

Uma alternativa que poderia ser mais explorada é o uso de ferramentas visuais, como programação por blocos, para reduzir a exigência nas habilidades ligadas à tecnologia da informação e comunicação, consequentemente permitindo que profissionais da área de aplicação possam usar ferramentas de visão computacional.

Programação por blocos é uma forma de criar programas que já mostrou que facilita/aceleira o desenvolvimento de aplicativos. É fácil de usar, razoavelmente intuitivo, pode inclusive ser usado para ensinar crianças a programar - talvez o uso mais popular da programação por blocos. Esta forma de programar tem suas limitações: à medida que a quantidade de blocos aumenta a ferramenta perde a comodidade pela poluição visual e quantidade e amplitude de movimentos físicos (arrastar objetos na tela) necessários para construir os programas. Ainda assim, com as escolhas adequadas, pode ser uma ferramenta útil para o desenvolvimento de aplicações.

App Inventor (POKRESS, VEIGA, 2013) é um dos mais conhecidos ambientes de programação em blocos. Desenvolvido pelo MIT, busca facilitar o desenvolvimento de aplicativos que têm como base a linguagem Java. Esta plataforma faz uso do Blockly, uma biblioteca que adiciona um código representado por blocos que formam expressões lógicas quando anexados uns aos outros. OpenCV (Bradski, 2000) significa Open Source Computer Vision Library , é uma biblioteca de software aberto para visão computacional e aprendizado de máquina (openCV, 2019). Originalmente desenvolvida na linguagem C/C++, por ser Open Source e ser constantemente otimizada, fez com que outras formas de aplicações com diferentes linguagens pudessem utilizá-la, tendo inclusive suporte ao Android. Dessa forma, possibilitou diversos avanços na pesquisa em visão computacional. Embora esteja disponível para ser incorporada às aplicações feitas no Android Studio, IDE oficial e profissional de aplicações Android, a biblioteca OpenCV não oferece suporte para o App Inventor e nem para o appybuilder. Há projetos que tentaram implementar funcionalidades semelhantes, mas de

forma limitada e desatualizada.

Objetivos

1. Criar uma ou mais extensões de alguma ferramenta de programação por blocos que permitam a pessoas pouco experientes em linguagens de programação utilizar técnicas de Visão Computacional.
2. Divulgar e documentar o desenvolvimento de extensões e os dados coletados com o intuito de atrair desenvolvedores e interessados da área de computação e de áreas de aplicação.
3. Introduzir o aluno de graduação da USP em uma linha de pesquisa nas Áreas de Visão Computacional, Inteligência Artificial e Construção de Aplicativos.

Métodos

Trata-se do desenvolvimento de componentes de software utilizando metodologia ágil. Nas primeiras semanas o estudante conhecerá colegas em diferentes estágios do seu trabalho de Iniciação Científica, que podem ser fontes de informação ou facilitadores para seu projeto. Nas semanas subsequentes, o estudante desenvolverá suas atividades recebendo metas semanalmente à medida que cumpre as metas anteriores, em um ciclo de desenvolvimento. Ao final espera-se entregar todos os resultados.

Resultados

Substituição de OpenCV por BoofCV.

Compilação de ImageCV.

Conteúdo detalhado (documentos/ExtensaolImageCV.md)

Uso da extensão - Construção de app para celular usando ImageCV e App Inventor.

Conteúdo detalhado (README.md)

Sumário de resultados - Indicadores de avaliação

1. Site documentando o projeto (<https://github.com/camilabezerril/ImageCV/tree/master>)
2. Entrega do primeiro protótipo no fim de Janeiro (appcv.ImageCV.aix);
3. Submissão de resumo ao WICSI no fim de Fevereiro;
4. Entrega de relatório intermediário no fim de Março (DocumentacaoProjeto.pdf);
5. Entrega da última versão da extensão em Agosto (appcv.ImageCV.aix);
6. Entrega do relatório final em Agosto (relatorio.md);

7. Submissão do resumo ao SIICUSP em Outubro.

Discussão e conclusão

A versão do BoofCV utilizada em ImageCV é 0.27, por questão de compatibilidade entre ambientes de desenvolvimento. A versão atual de BoofCV é 0.36. A documentação da versão 0.27 não foi encontrada, o que dificultou o uso das funções de reconhecimento de polígonos.

Referências

BRADSKI, G. (2000) - The OpenCV Library. Dr. Dobb's Journal of Software Tools . OpenCV (2019) - About OpenCV - < <https://opencv.org/about/> (<https://opencv.org/about/> %E2%80%8B) > Acessado em 26 de maio de 2019. POKRESS, Shaileen Crawford; VEIGA, José Juan Dominguez (2013) - MIT App Inventor: Enabling Personal Mobile Computing - PRoMoTo 2013 Proceedings (arXiv:1309.5509) October 2013.

Lista de divulgação dos resultados, quando for o caso.

ImageCV

Desenvolvimento de extensão para visão computacional no ambiente de programação por blocos App Inventor

Esta documentação faz parte dos entregáveis do projeto de Iniciação Científica XXX/2019.

Fique nesta página se quiser criar no app inventor um app de celular que usa a extensão. Caso seja a primeira vez que usará App Inventor, sugerimos ler, ou seguir nosso tutorial para criar seu primeiro app usando App Inventor (Documentos/primeiroApp/primeiroApp.md)

Navegue para Como criar extensões do app Inventor (Documentos/ComoCriarExtensao.md) para ver o que foi feito para criar a extensão (setup de IDE, inclusão de bibliotecas, ...).

Navegue para Como a extensão foi codificada (Documentos/ExtensaolImageCV.md) para ver como chegou-se à esta versão do código (escolha de tipos de dados, escolhas de funcionamento dos blocos, ...).

Navegue para Apps de teste (testes/README.md) para baixar os executáveis (apk) e os fontes (aia) dos apps de teste.

Navegue para Como github foi usado neste projeto (Documentos/sobreGit/README.md) para ver que recursos de github foram usados neste projeto. Pode ser um bom guia para começar a usar **git** e **github**.

Apresentação (contexto)

App Inventor (<https://appinventor.mit.edu/>) é uma ferramenta de programação baseada em blocos (*block-based programming* (https://en.wikipedia.org/wiki/Visual_programming_language)) para criar *apps* para Android (https://www.android.com/intl/pt-BR_br/). Numa generalização grosseira, esses *apps* podem ser executados em telefones celulares, tablets e outros dispositivos que usam Android como sistema operacional.

Ferramentas como esta tornaram-se conhecidas por seu sucesso no uso em ensino de programação para crianças. O que também criou o estigma de *ser coisa para criança*.

A técnica de programação baseada em blocos favorece a criação modular de componentes, acelera o desenvolvimento de *apps*, seja na fase de protótipo, seja na fase de produto, permite que profissionais com formação diversas das formações relacionadas à computação possam testar idéias facilmente e, mesmo para programadores, permite que se ocupem menos com detalhes da linguagem.

A versão web do App Inventor (ai2.appinventor.mit.edu/) não requer instalação de programa no computador local. O acesso é gratuito mediante cadastro.

Tutoriais sobre App Inventor, no site oficial em inglês, podem ser assistido aqui (<https://appinventor.mit.edu/explore/ai2/beginner-videos>).

Há alguns tutoriais em português:

androidpro (usa a versão desktop - instalada no computador) (<https://www.androidpro.com.br/blog/desenvolvimento-android/app-inventor/>)

appinventorbrasil (**parece ser** acesso gratuito, mediante cadastro)
(<https://appinventorbrasil.tech/cursos/app-inventor-2-iniciantes>)

Criar um app no App Inventor consiste em entrar no app inventor web (ai2.appinventor.mit.edu/), criar um projeto (para isso é necessário cadastrar-se), no modo *design* arrastar os componentes que se deseja usar para dentro do dispositivo (celular/tablet), entrar no modo *blocks* escolher, dos componentes, que bloco usar, arrastá-los para dentro do canvas e encaixá-los uns nos outros para que o conjunto se comporte como desejado. Ao final, ou quando uma versão testável puder ser criada, escolher no menu *compile*, uma das opções (a mais simples é compilar, gerar um QR-code e ler com um leitor previamente instalado no dispositivo), baixar o instalador do *app* (geralmente um arquivo com extensão .apk), instalar e usar/testar.

Uma **extensão** do App Inventor é um componente criado por um desenvolvedor independente.

ImageCV é um componente do App Inventor que, dada uma imagem, por exemplo uma foto, permite:

1. Marcar determinadas cores;
2. Localizar elipses;
3. Localizar polígonos, definindo a quantidade de vértices desejada;
4. Combinar a marcação de cores com a localização de formas.

Imagens que exemplificam usos simples

Há muitas formas de usar e combinar as funcionalidades do ImageCV. Documentar e exemplificar exaustivamente geraria muita documentação de pouca utilidade.

Supondo que o seu processo de avaliação da utilidade de ImageCV seja:

1. Ver o que ImageCV faz: o que usa como entrada e o que apresenta como saída;
2. Verificar se há exemplos que se aproximam do uso que deseja;
3. Experimentar quanto tempo e conhecimento são necessários para começar a usá-lo;
4. Aprofundar-se nos usos e na documentação para chegar ao uso desejado;

Apresentamos abaixo as imagens usadas nos testes que fizemos e os resultados.

Nas seções seguintes apresentamos como instalar a extensão no App Inventor, como construir, rapidamente, um app usando algumas funcionalidades de ImageCV, a documentação de referência do ImageCV. Lembre-se que há apps para teste do ImageCV (testes/README.md).

(**nota:** um pesquisador da UNICAMP propõe criar uma versão do App Inventor traduzida para português (<https://appinventor.mit.edu/explore/blogs/josh/2016/01/mit-0>)

(**nota:** a interface web tem suporte para Português do Brasil)

(A falta de um tutorial em português pode ser barreira para seu uso com crianças brasileiras.)

(<https://cacm.acm.org/magazines/2019/8/238340-block-based-programming-in-computer-science-education/fulltext> (<https://cacm.acm.org/magazines/2019/8/238340-block-based-programming-in-computer-science-education/fulltext>) https://en.scratch-wiki.info/wiki/Block-Based_Coding (https://en.scratch-wiki.info/wiki/Block-Based_Coding)

)

(**nota:** para cortar as imagens e diminuir seu tamanho usei imageMagic, segundo <https://itectec.com/ubuntu/ubuntu-cropping-images-using-command-line-tools-only/> (<https://itectec.com/ubuntu/ubuntu-cropping-images-using-command-line-tools-only/>)

```
fabio@fabio-13Z940-G-BK71P1:~/Documentos/Camila/screenshots-primeiroApp$  
identify Captura\ de\ tela\ de\ 2020-06-14\ 08-47-54.png  
Captura de tela de 2020-06-14 08-47-54.png PNG 1080x1920 1080x1920+0+0 8-bit sRGB 1.974MB 0.000u 0:00.000
```

```
fabio@fabio-13Z940-G-BK71P1:~/Documentos/Camila/screenshots-primeiroApp$  
convert Captura\ de\ tela\ de\ 2020-06-14\ 10-45-03.png -quality 50 -crop  
1080x1000+50+0 17.jpg
```

)

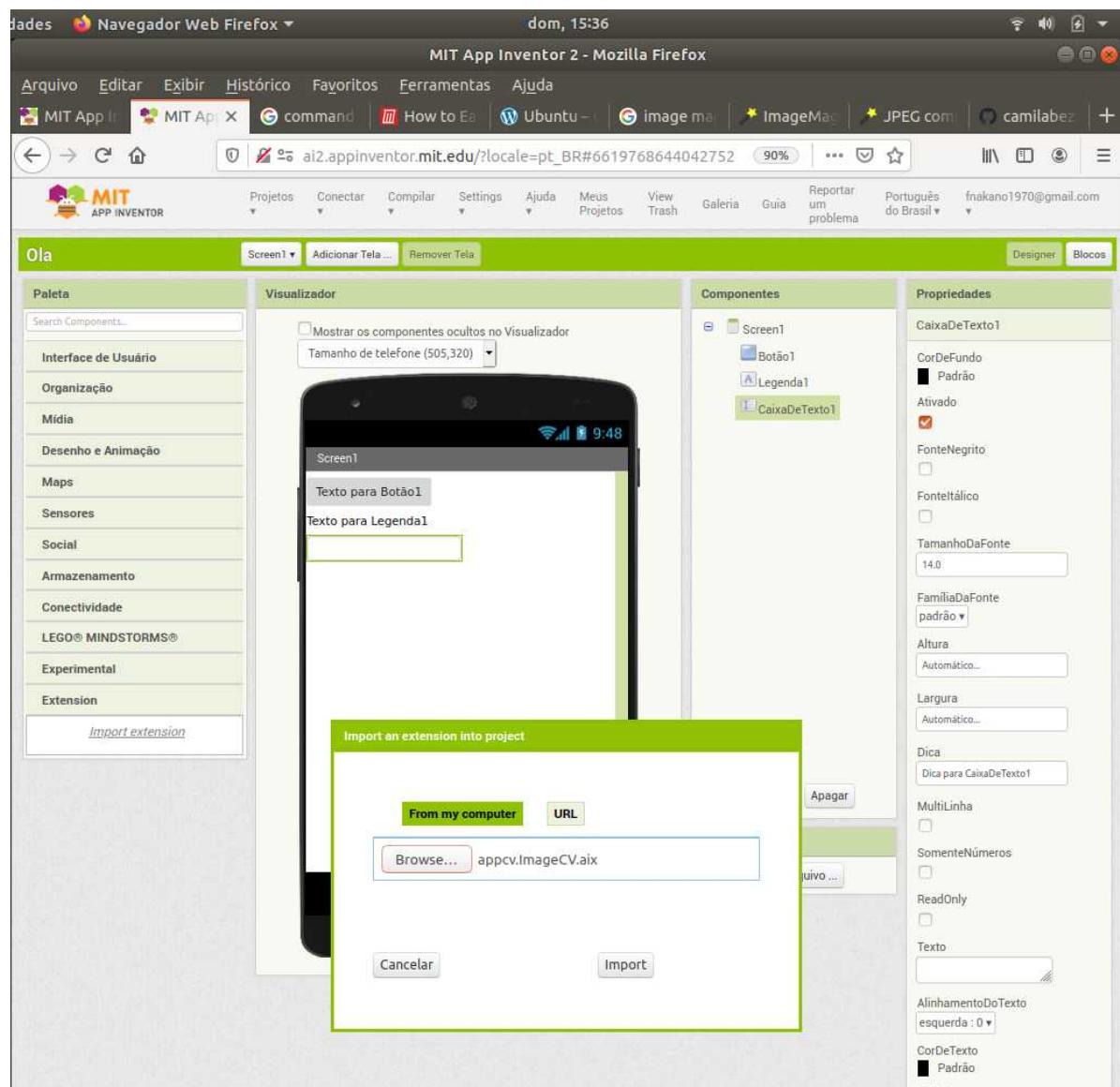
Como instalar a extensão

Baixe a extensão ImageCV aqui ([appcv.ImageCV.aix](#)), guarde onde ela foi armazenada.

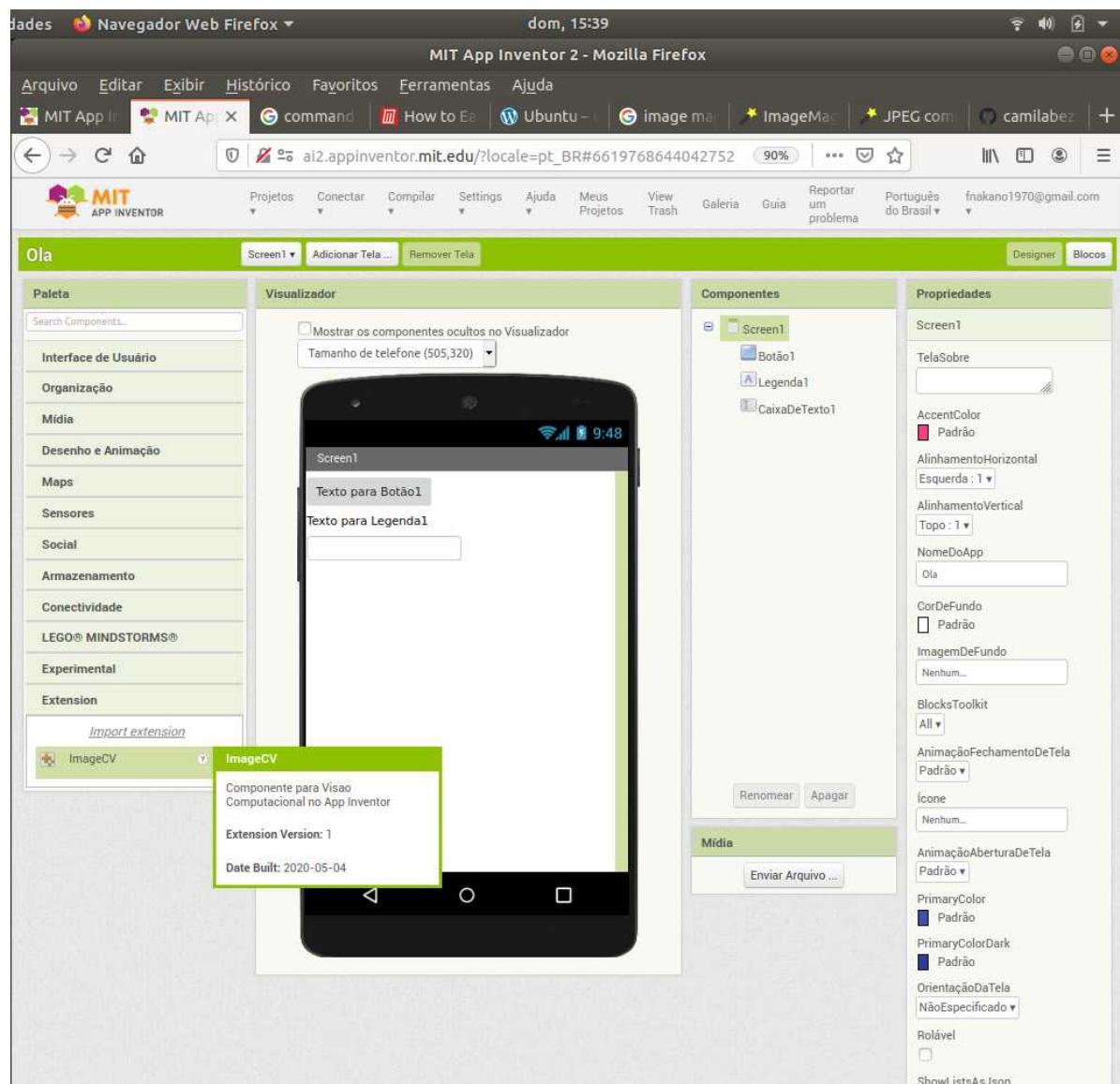
No seu projeto do App Inventor, no quadro da esquerda na aba extension clique no link *Import extension*

Uma caixa de mensagem se apresenta, clique em Browse, selecione o arquivo *appcv.ImageCV.aix* baixado no passo X.

ImageCV



Deve aparecer na aba a extensão ImageCV.



Como usar a extensão em um app simples

Referência dos blocos de ImageCV

Por que HSV ao invés de RGB.

Em Visão Computacional, um componente que identifica uma determinada cor, recebe a informação sobre a cor a analisar e aplica uma fórmula. O resultado da aplicação da fórmula indica se a cor analisada é a determinada cor ou se a cor analisada não é a determinada cor.

Para manter o ImageCV simples para o programador que usa App Inventor e simples para desenvolvedores que forem aperfeiçoá-lo, os parâmetros passados para os blocos são números e não fórmulas. Acredite, construir boas fórmulas pode não ser fácil e o programador que usa App Inventor ficaria encarregado disso.

RGB e HSV são duas formas de codificar cores. No RGB, as cores são decompostas em

componentes de cor, sendo estas as intensidades de vermelho (R), verde (G) e azul (B). No ImageCV, as intensidades de R, G e B vão de 0 a 255. Por exemplo:

R	G	B	Nome da cor
0	0	0	preto
255	255	255	branco
255	0	0	vermelho intenso
120	0	0	vermelho médio
255	83	0	laranja-avermelhado (ou vermelho-alaranjado?)
255	203	203	um tom claro de rosa
255	0	255	magenta
255	255	0	amarelo
0	200	0	um tom claro de verde

Seguindo a idéia de manter o ImageCV simples, na mais complexa das alternativas, permitiríamos valores mínimos e máximos de R, G e B. Desta forma, se o programador quisesse um vermelho, poderia usar algo como `min={200,0,0} max={255, 20, 20}`. Esta escolha deixaria de fora tanto os vermelhos mais escuros quanto os rosas mais claros, os vermelhos alaranjados, os vermelhos azulados,...

O ponto é que usar RGB e colocar mínimo e máximo para definir cores leva a resultados corretos (dada a codificação), mas difíceis para pessoas entenderem, considerando a forma como percebem e categorizam cores.

Na codificação HSV, H corresponde à tonalidade, S à saturação e V a "claridade". H codifica tons: vermelhos, amarelos, verdes,... S codifica quão pigmentada é. Valores altos correspondem a cores intensas, vivas. V codifica se há pouca luz ou luz suficiente. Desta forma, o vermelho tem uma faixa de H, vermelhos pouco saturados (rosa) têm S baixos, vermelhos muito saturados (talvez como o nariz de um palhaço) têm S altos. V tem pouca influência na discriminação de cor.

A biblioteca usada em ImageCV para processamento de cores e formas é BoofCV. A explicação sobre HSV em BoofCV está em (<http://boofcv.org/javadoc/boofcv/alg/color/ColorHsv.html>) (<http://boofcv.org/javadoc/boofcv/alg/color/ColorHsv.html>)

(**nota** A Wikipedia (https://en.wikipedia.org/wiki/HSL_and_HSV) explica com mais detalhes.)

(**nota**: os sensores das câmeras, em situações com muita luz, tendem a levar as cores para o branco. Diz-se que o sensor saturou, ou que as cores estão "estouradas". A codificação HSL

parece lidar melhor com essa situação que a codificação HSV.)

stepHSV

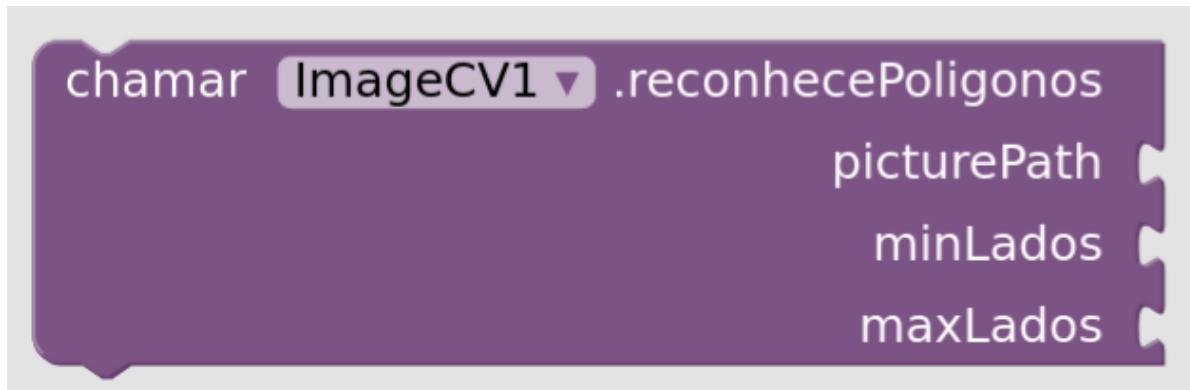


picturePath: localização do arquivo de imagem. HSMin: valores mínimos de H e S. HSMax: valores máximos de H e S.

Retorna em altImagePath a localização do arquivo de imagem segmentado por cor.

baseado em (https://boofcv.org/index.php?title=Example_Color_Segmentation
https://boofcv.org/index.php?title=Example_Color_Segmentation)

reconhecePoligonos



picturePath: localização do arquivo de imagem. minLados: quantidade mínima de lados dos polígonos a identificar (mínimo=3, máximo=20) maxLados: quantidade máxima de lados dos polígonos a identificar (mínimo=3, máximo=20)

O desempenho ótimo é conseguido quando os polígonos são pretos e planos em fundo branco, com iluminação uniforme, sem sombras.

Retorna em altImagePath a localização do arquivo de imagem com os polígonos identificados contornados em vermelho.

Retorna em nPoligonos a quantidade de polígonos encontrada.

Retorna em listaDePoligonos uma lista contendo nPoligonos sub-listas. Cada sub-lista contém a coordenadas dos vértices dos polígonos em ordem: [x1, y1, x2, y2, x3, y3, ..., xn, yn].

reconheceElipses



```
chamar ImageCV1 .reconheceElipses
picturePath
```

picturePath: localização do arquivo de imagem.

O desempenho ótimo é conseguido quando os polígonos são pretos e planos em fundo branco, com iluminação uniforme, sem sombras.

Retorna em altImagePath a localização do arquivo de imagem segmentado por cor.

Retorna em nElipses a quantidade de elipses encontrada.

Retorna em listaDeElipses uma lista contendo nElipses sub-listas. Uma sub-lista contém: [xcentro, ycentro, "Não Especificado", diâmetro_médio].

converteRGBtoHSV



```
chamar ImageCV1 .converteRGBtoHSV
RGB
```

RGB

getAltImagePath



```
ImageCV1 . getAltImagePath
```

Contém a localização do arquivo de imagem que contém as cores segmentadas ou as elipses ou os polígonos localizados, dependendo do método que foi utilizado.



```
ImageCV1
```

getCaractElipses

ImageCV1 ▾ . getListadeElipses ▾

getCaractPoligonos

ImageCV1 ▾ . getListadePoligonos ▾

getHfromRGBtoHSV

ImageCV1 ▾ . getHfromRGBtoHSV ▾

getSfromRGBtoHSV

ImageCV1 ▾ . getSfromRGBtoHSV ▾

getVfromRGBtoHSV

ImageCV1 ▾ . getVfromRGBtoHSV ▾

getListadeElipses

getnElipses

ImageCV1 ▾ . getnElipses ▾

getnPoligonos



setCaractElipses

setCaractPoligonos

setnElipses



setnPoligonos



```
@DesignerComponent(version = 1,
    category = ComponentCategory.EXTENSION,
    description = "Componente para Visao Computacional no App Inventor",
    nonVisible = true,
    iconName = "images/extension.png")

    @SimpleFunction(description = "Substitui os tons de uma cor por preto
por meio do sistema HSV [FORMATO: 0-6.28, 0-1.0]")
    public void stepHSV(String picturePath, String HSmin, String HSmax){

        @SimpleFunction(description = "Converte RGB em HSV [FORMATO:x,x,x]")
        public void converteRGBtoHSV(String RGB) {

            @SimpleFunction(description = "Reconhece elipses em uma imagem")
            public void reconheceElipses(String picturePath){

                @SimpleFunction(description = "Reconhece polígonos em uma imagem, min
Lados >= 3 e maxLados <= 20")
                public void reconhecePoligonos(String picturePath, int minLados, int
maxLados) {
```

Como criar extensões do App Inventor

Visão geral

Extensões do App Inventor são programas que acrescentam alguma funcionalidade, em geral, não disponível nos blocos já existentes e nem atingível pela composição de blocos já existentes.

Estas extensões não podem ser componentes da interface gráfica com o usuário (GUI). Por isso, às vezes são chamadas *invisíveis*.

A extensão do nome do arquivo que contém uma extensão do App Inventor é *aix*. É algo como *nome.aix*.

Como desenvolver

As extensões do App Inventor são mantidas localmente: o ambiente de desenvolvimento é instalado e executado no computador do desenvolvedor (ié localmente) e o código-fonte fica armazenado neste. Por isso, é necessário fazer alguns ajustes no computador.

Um conjunto de arquivo-fonte é necessário para definir objetos para comunicação entre a nova extensão e os blocos já existentes no App Inventor. Ele é disponibilizado através do github (github.com). A forma mais fácil de obtê-lo é clonando o repositório do App Inventor (<https://github.com/mit-cml/appinventor-sources.git>).

Em seguida, há uma sistematização do procedimento para obter uma extensão.

Configurando o Ambiente

Para possibilitar o desenvolvimento da extensão, as seguintes ferramentas devem ser instaladas:

- **Java JDK**
 - Download aqui (<https://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html>)
- **Git**
 - Download aqui (<https://git-scm.com/>)
- **Apache Ant**
 - Download aqui (<https://ant.apache.org/bindownload.cgi>)

Apache Ant é uma biblioteca Java e ferramenta de linha de comando essencial que assiste na

compilação dos arquivos da extensão em conjunto com os do App Inventor.

Após a instalação das ferramentas anteriores, as variáveis de ambiente devem ser acertadas nas propriedades do sistema no painel de controle do Windows:

Variável	Valor
ANT_HOME	F:\apache-ant-1.10.7 (Pasta do Apache)
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_111 (Pasta do Java)
ANT_OPTS	-Xmx256M
_JAVA_OPTIONS	-Xmx1024m
CLASSPATH	%ANT_HOME%\lib;%JAVA_HOME%\lib
Path	%ANT_HOME%\bin;%JAVA_HOME%\bin

Por meio do Git, os arquivos do App Inventor serão clonados:

1. Selecione o local dos arquivos, no meu caso o local é: C:\
2. Clique com o botão direito e em Git bash here
3. Na linha de comando digite: git clone <https://github.com/mit-cml/appinventor-sources.git>

Localização da extensão

Abrindo a pasta **appinventor**, dentro da pasta recém-clonada, como um projeto em um Ambiente de Desenvolvimento Integrado (IDE), as extensões por padrão devem ser colocadas no caminho: appinventor\components\src\com\google\appinventor\components\runtime

A IDE que utilizei para o desenvolvimento da extensão é a IntelliJ (<https://www.jetbrains.com/pt-br/idea/>) da JetBrains.

O Código-fonte da extensão

Importação de pacotes/classes do App Inventor

Criada a classe da extensão no local adequado, os *imports* mais comuns para funcionalidade do App Inventor, que definem a forma como os blocos são vistos e utilizados pelo usuário, mensagens de erro, uso de mídia, entre outros, são:

```

import android.Manifest;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;
import com.google.appinventor.components.runtime.errors.IllegalArgumentExceptionError;
import com.google.appinventor.components.runtime.util.AnimationUtil;
import com.google.appinventor.components.runtime.util.ErrorMessages;
import com.google.appinventor.components.runtime.util.HoneycombUtil;
import com.google.appinventor.components.runtime.util.MediaUtil;
import com.google.appinventor.components.runtime.util.SdkLevel;
import com.google.appinventor.components.runtime.util.ViewUtil;
import com.google.appinventor.components.annotations.UsesLibraries;
import com.google.appinventor.components.runtime.*;

```

Note que há inclusão do *import* **UsesLibraries**, necessário quando há a necessidade de uso de bibliotecas externas informando posteriormente as bibliotecas essenciais que serão utilizadas pela extensão.

Detalhes da extensão

Há uma seção que informa os detalhes do código em desenvolvimento para o App Inventor:

```

@DesignerComponent(version = 1,
    category = ComponentCategory.EXTENSION,
    description = "Componente para Visão Computacional no App Inventor"
    nonVisible = true,
    iconName = "images/extension.png")
@SimpleObject(external = true)

```

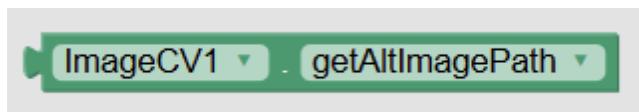
A categoria da classe deve ser, por padrão, EXTENSION. As extensões por padrão, não podem ser componentes visíveis (design do App Inventor) para o usuário, atuando apenas na parte de programação por blocos

Algumas propriedades do App Inventor

Os métodos que poderão ser vistos pelo usuário para a programação por blocos em geral possuem propriedades que aparecerem antes da inicialização destes e definem as formas que poderão ser utilizados. Tais como:

@SimpleProperty define o método seguinte como um bloco de propriedade simples de alguma outra função ou método, portanto atua como um parâmetro, como no exemplo a seguir:

```
@SimpleProperty(category = PropertyCategory.BEHAVIOR)
public String getAltImagePath() {
    return altImagePath;
}
```



@SimpleFunction define o método seguinte como um bloco de função que receberá parâmetros a serem definidos no mesmo.

```
@SimpleFunction(description = "Reconhece elipses em uma imagem")
public void reconheceElipses(String picturePath){
    //...
}
```



Compilando a extensão

Para compilar a extensão, é necessário clicar com o botão direito e em seguida em `Git bash here` dentro da pasta appinventor para entrar na linha de comando do git.

Há dois comandos essenciais:

- `ant clean` limpa compilações anteriores (sempre utilizar antes de `ant extensions`)
- `ant extensions` compila arquivos e gera arquivo aix da extensão

Após a conclusão correta da compilação (sinalizada pela mensagem: `BUILD SUCCESSFUL`), um arquivo aix é gerado e está localizado na pasta `appinventor\components\build\extensions` podendo então ser aplicado a qualquer projeto do App Inventor por meio da seção extension no ambiente de design deste.

Como a extensão foi codificada

A extensão ImageCV está sendo desenvolvida com base na biblioteca de visão computacional BoofCV (https://boofcv.org/index.php?title=Main_Page), escolhida por não utilizar bibliotecas “nativas” codificadas em linguagem diferente de Java, ser Open Source, estar em constante atualização, e ainda por ter suporte ao Android.

As seções deste documento são:

1. Pacotes do BoofCV
2. Funcionamento da extensão
 1. Funcionalidade básica
 2. Funções relacionadas à visão computacional

Pacotes do BoofCV

A biblioteca BoofCV (Atualmente na versão 0.36) é disponibilizada de duas formas: uma delas apresenta as dependências colocadas de forma a serem facilmente adicionadas pelas IDE's por meio de comandos. Já a outra é através de arquivos jar, que devido à plataforma para o qual a extensão é desenvolvida (App Inventor), é a forma utilizada para aplicação da biblioteca na extensão.

A versão da biblioteca selecionada é a 0.27 (Baixe aqui (<https://sourceforge.net/projects/boofcv/files/v0.27/boofcv-v0.27libs.zip/download>)), podendo ainda ser modificada para a 0.28. As versões mais recentes não são utilizadas devido à incompatibilidade do App Inventor com a biblioteca, o primeiro tendo suporte para o Java 7, já a segunda desenvolvida para o Java 8-11 a partir da versão 0.29.

As demais versões (<https://boofcv.org/index.php?title=Download>) estão disponíveis no site da própria biblioteca.

Os pacotes utilizados pela extensão atualmente são:

- boofcv-android.jar
- boofcv-ip.jar
- ddogleg.jar
- georegression.jar
- boofcv-feature.jar
- ejml-core.jar
- ejml-ddense.jar
- ejml-fdense.jar
- ejml-simple.jar

Configuração dos pacotes

Após serem baixados, os pacotes que serão utilizados devem ser colocados na pasta `appinventor\lib`. Estes podem ser colocados em uma subpasta com nome a ser escolhido.

Para que seja compilado devidamente, o arquivo `build.xml` (`../build.xml`), localizado dentro da pasta `appinventor\components`, deve ser modificado.

A seguinte formatação deve ser colocada na seção `CopyComponentLibraries target` (Há um comentário com uma indicação: `<!-- Add extension libraries here -->`) para cada pacote adicionado à pasta `lib` que será utilizado:

```
<copy toFile="${public.deps.dir}/nomeSimplificadoDoArquivoJar.jar"
      file="${lib.dir}/SubpastaDaBibliotecaNoLib/nomeDoArquivoJar.jar" />
```

Se estiver correto, após a compilação os pacotes serão copiados da pasta `lib` para a pasta `appinventor\build\components\deps`.

Por último, é necessário alterar o arquivo da classe da extensão de forma a informar quais bibliotecas serão utilizadas durante a extensão, etapa na qual já está feita para os arquivos `jar` que são atualmente utilizados. Debaixo dos detalhes da extensão, a seguinte formatação deve estar colocada:

```
@UsesLibraries(libraries = "library1.jar," + "library2.jar," + ... + "libraryN.jar")
```

Funcionamento da extensão

Funcionalidade básica

Métodos acessores

Logo ao início está a parte do código que disponibiliza para o usuário as informações geradas pela extensão, possibilitando também que o mesmo possa modificar algumas delas.

Por exemplo:

```
/**
 * Retorna imagem alterada pela extensao
 */
@SimpleProperty(category = PropertyCategory.BEHAVIOR)
public String getAltImagePath() {
    return altImagePath;
}
```

Imagen após ser fotografada

O método converteDrawableToBitmap trata a imagem assim que a foto é tirada pelo componente da câmera do App Inventor e seu caminho se torna disponível para uso.

O caminho da imagem é recebido e um drawable é gerado a partir disto, utilizando uma classe do próprio App Inventor (MediaUtil) que trata as variantes de uma imagem. Em seguida um bitmap em escala é criado, evitando possíveis problemas gerados pelas imagens rotacionadas etc. e retornado.

A conversão de drawable para bitmap ocorre para que os processos envolvendo visão computacional sejam aplicados.

Salvando imagem alterada pela extensão

No método salvaAltImage o bitmap que passou por algum processo e foi alterado é comprimido, convertido para uma jpeg e salvo na galeria do celular por meio de um objeto file.

Por fim, nesse mesmo método, o caminho da imagem alterada é salvo para posterior uso, como exibição desta no aplicativo, entre outros.

Todas as imagens que passam por processos da extensão são salvas no mesmo caminho: pictures/ImageCV

Funções relacionadas à visão computacional

Note que muitas classes da biblioteca BoofCV utilizam o tipo BufferedImage para processar as imagens, porém esta classe não é suportada pelo Android. Alternativamente, a biblioteca apresenta classes que possibilitam a conversão de bitmap para tipos de imagens em que a mesma trabalha. Estas classes estão presentes no pacote jar boofcv-android-0.27.

Marcando cores

Como uma forma de ampliar as informações adquiridas através de uma imagem, é possível a marcação de determinadas cores com o método stepHSV. Neste método a imagem é convertida de RGB para o HSV por meio do BoofCV. Este formato possibilita uma maior abrangência na captura de cores e uma manipulação das informações mais facilmente. O usuário especifica a cor a ser identificada (em formato HSV). Se esta cor for encontrada, os pixels relacionados têm suas cores alteradas para o preto (Cor no qual o reconhecimento de figuras geométricas também se torna possível) e o resto da imagem é alterada para a cor branca.

Esta função pode ser chamada diversas vezes para marcar diversas cores, não sendo assim necessário especificar uma cor que estaria precisamente na imagem.

Reconhecimento de elipses

O reconhecimento de elipses ocorre através do método reconheceElipses. Este pode ser feito

de duas formas, utilizando do reconhecimento de cores em conjunto ou apenas com o método citado. Se há elipses de outras cores que não a preta que devem ser também analisadas, é recomendado que o reconhecimento de cores seja utilizado em conjunto para uma melhor análise.

Se agindo em conjunto com o reconhecimento de cores, é possível saber quais as cores primárias mais próximas em cada elipse encontrada e ainda, encontrar apenas elipses de uma determinada cor.

Além disso, outras informações, como as coordenadas do centro da elipse e o seu perímetro aproximado, são fornecidas.

Reconhecimento de polígonos

O reconhecimento de polígonos é feito através dos métodos: reconhecePoligonos, processaPoligonos, renderPolygon. Este reconhecimento atua da mesma forma que o de elipses, podendo ser utilizado em conjunto com o reconhecimento de cores também. Tanto polígonos convexos como côncavos podem ser analisados. A análise pode ainda ser limitada para polígonos com um certo número de vértices ou ainda por uma determinada cor, ambos podendo ser especificados pelo usuário.

Além da informação da cor de cada polígono encontrado, também são fornecidas as coordenadas dos vértices dos polígonos.

Disponibilização das informações

As informações que passam pelo reconhecimento de figuras geométricas são disponibilizadas por meio de uma lista de listas que pode ser acessada como uma lista padrão do App Inventor.

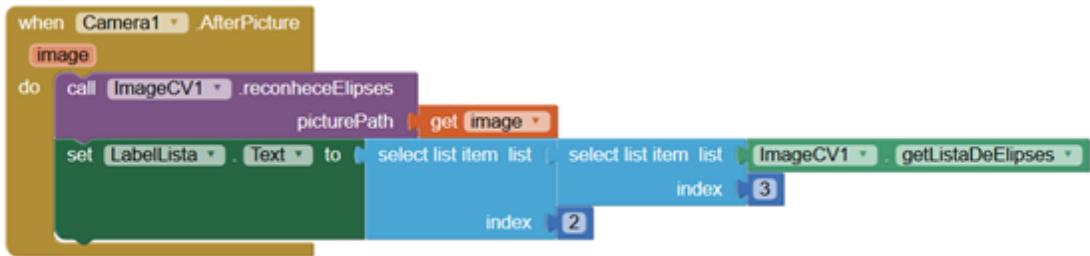
As elipses estão contidas em uma lista que possui listas com informações na seguinte ordem: coordenada x do centro, coordenada y do centro, cor da elipse, perímetro da elipse.

Já os polígonos estão contidos em uma lista que possui listas com informações na seguinte ordem: cor do polígono, coordenada x do vértice 1, coordenada y do vértice 1, coordenada x do vértice 2, coordenada y do vértice 2...

As listas podem ser acessadas e usadas de diversas formas, como por exemplo:
ListaDeElipses[3].cElipse[2] - Acessando o Cy da Elipse 3 (As iterações do AppInventor começam em 1)

Em programação por blocos, isso seria o equivalente a:

Como a extensão foi codificada



Para conter essas informações foi utilizado a lista primitiva do App Inventor: **YailList**. O código-fonte desta está no caminho: \appinventor-sources\appinventor\components\src\com\google\appinventor\components\runtime\util\YailList.java

Além disso, o número de elipses e de polígonos encontrados estão disponíveis por meio das variáveis **nElipses** e **nPoligonos**.

Construir o primeiro App para celular

Nesta página você vai aprender a criar um app/programa que pode ser executado em telefones e tablets que usam Android como sistema operacional.

Você vai desenvolver o app através da Internet, em um computador, usando o navegador. Nenhum programa adicional será necessário no computador.

No celular será necessário um leitor de QR-Code. Através dele, acessa-se o link para baixar seu app no celular quando você for testá-lo. Os passos são:

1. Criar uma conta no App Inventor Web ([Criar-uma-conta-no-App-Inventor-Web](#));
2. Criar um projeto ([Criar-um-projeto](#));
3. Colocar os componentes na tela ([Colocar-os-componentes-na-tela](#));
4. Definir as ações ([Definir-as-a%C3%A7%C3%B5es](#));
5. Enviar o app para o celular ([Enviar-o-app-para-o-celular](#)) e
6. Testar o app ([Testar-o-app](#)).

Criar uma conta no App Inventor Web.

A página de início é (<http://appinventor.mit.edu> (<http://appinventor.mit.edu>)). Logo no topo, clique em Create Apps!.

The screenshot shows the homepage of the MIT App Inventor website. At the top, there's a navigation bar with links for Arquivo, Editar, Exibir, Histórico, Favoritos, Ferramentas, and Ajuda. Below the navigation is a search bar with the URL appinventor.mit.edu. On the left, there's a logo featuring three colorful shapes (purple, green, yellow) and the text "MIT APP INVENTOR". On the right, there's a button labeled "Create Apps!" and a menu icon. The main banner features the text "With MIT App Inventor, anyone can build apps with global impact" over a background image of people working on computers. Below the banner is a statistics section showing active users and registered users from today, this week, and this month, along with the number of countries and apps built. A blue banner at the bottom announces the "MIT App Inventor Hackathon 2020".

Active Users today:	Active Users this week:	Active Users this month:	Registered Users:	Countries:	Apps Built:
27.1K	191.0K	637.7K	8.2M	195	34.0M

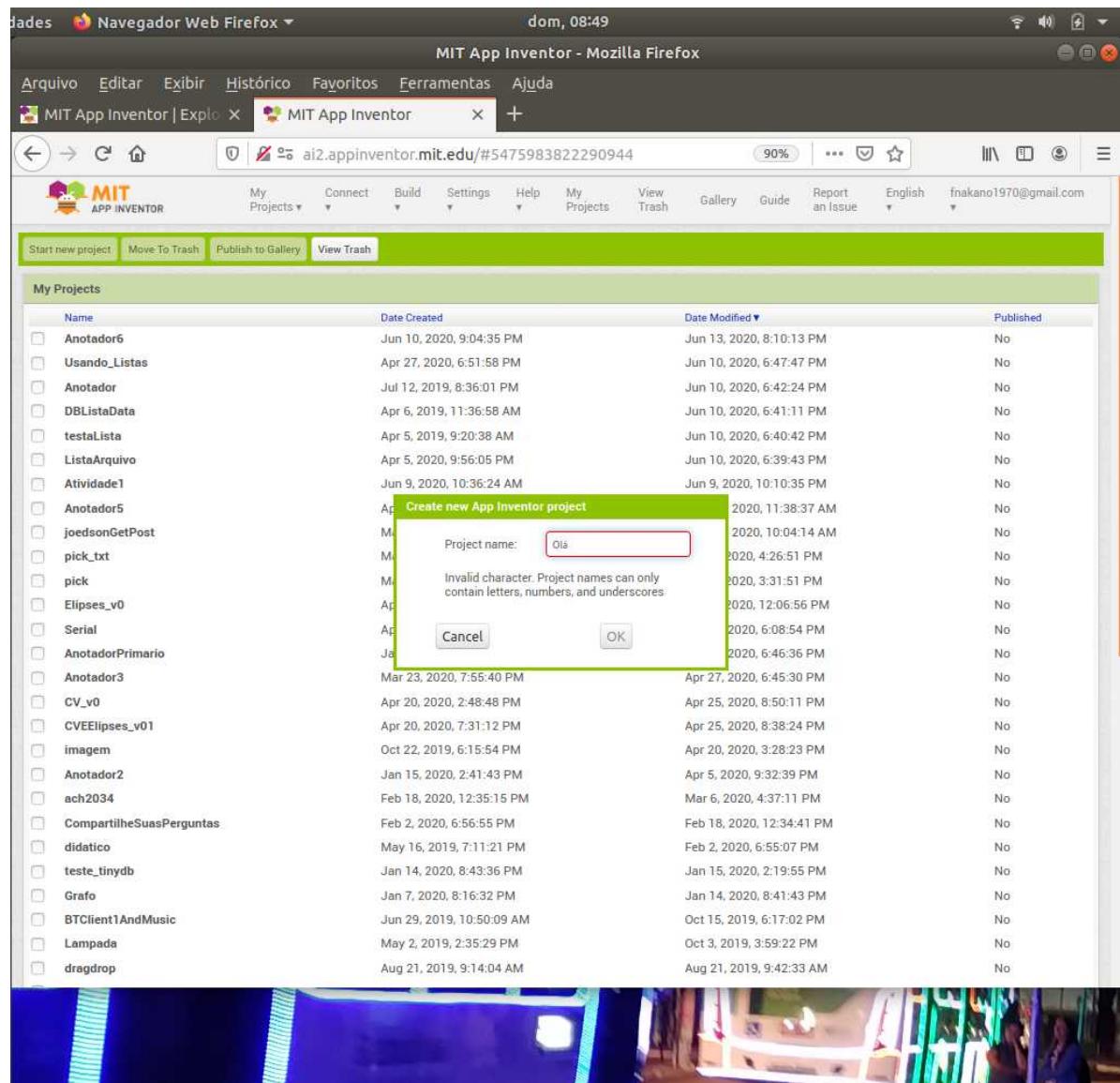
Announcing the MIT App Inventor Hackathon 2020, A Global App Hackathon for Good, from July 12-18. Sign up by June 19 to [participate](#).

Faça seu cadastro e entre na página para criar seus projetos.

Criar um projeto.

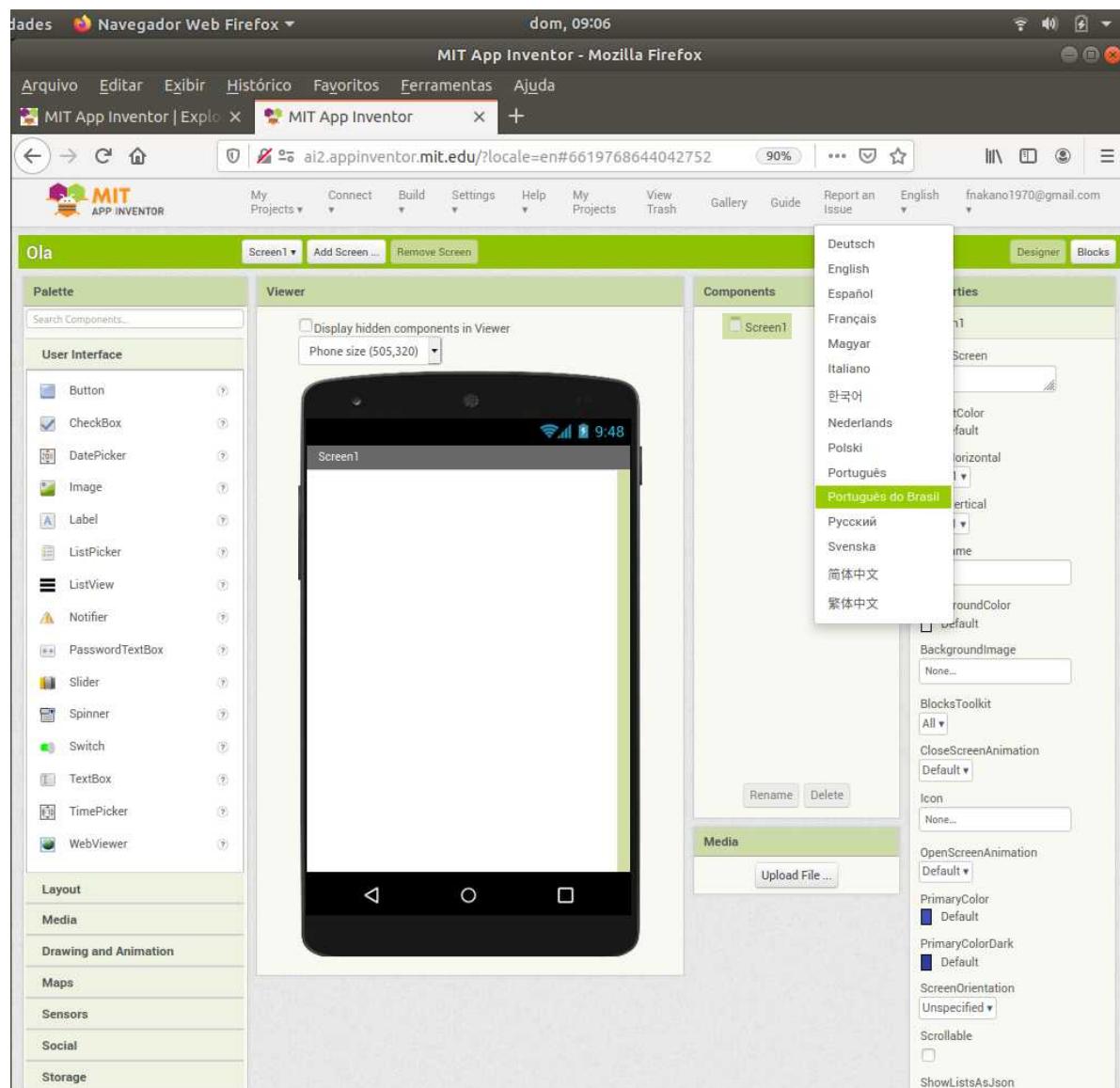
Em cima, ao lado do logo do App Inventor, há uma barra de menu. Com os menus é possível mudar de idioma, criar um novo projeto, trocar de projeto, criar o app e gerar um QR-Code para baixar e instalar o app no celular. Escolha criar um novo projeto e digite o nome do projeto na caixa de diálogo. Recomendamos o nome *Ola...* aparece o desenho de uma tela de celular.

Construir o primeiro App para celular



Se quiser, troque o idioma para Português do Brasil

Construir o primeiro App para celular



Colocar os componentes na tela.

Usando como referência o desenho da tela do celular, o quadro do lado direito apresenta as propriedades do componente selecionado - no caso a tela. Seu nome é Screen1, o nome do App é Ola.

No App Inventor, componentes são partes do app. Por exemplo, a tela, botões, caixas de texto, caixas de figuras, listas, etc.

Entre o desenho da tela e o quadro de propriedades fica a lista de componentes. No momento só há a Screen1. Nesta lista há botões para renomear e apagar o componentes.

No quadro do lado esquerdo ficam os componentes que podem ser usados no app.

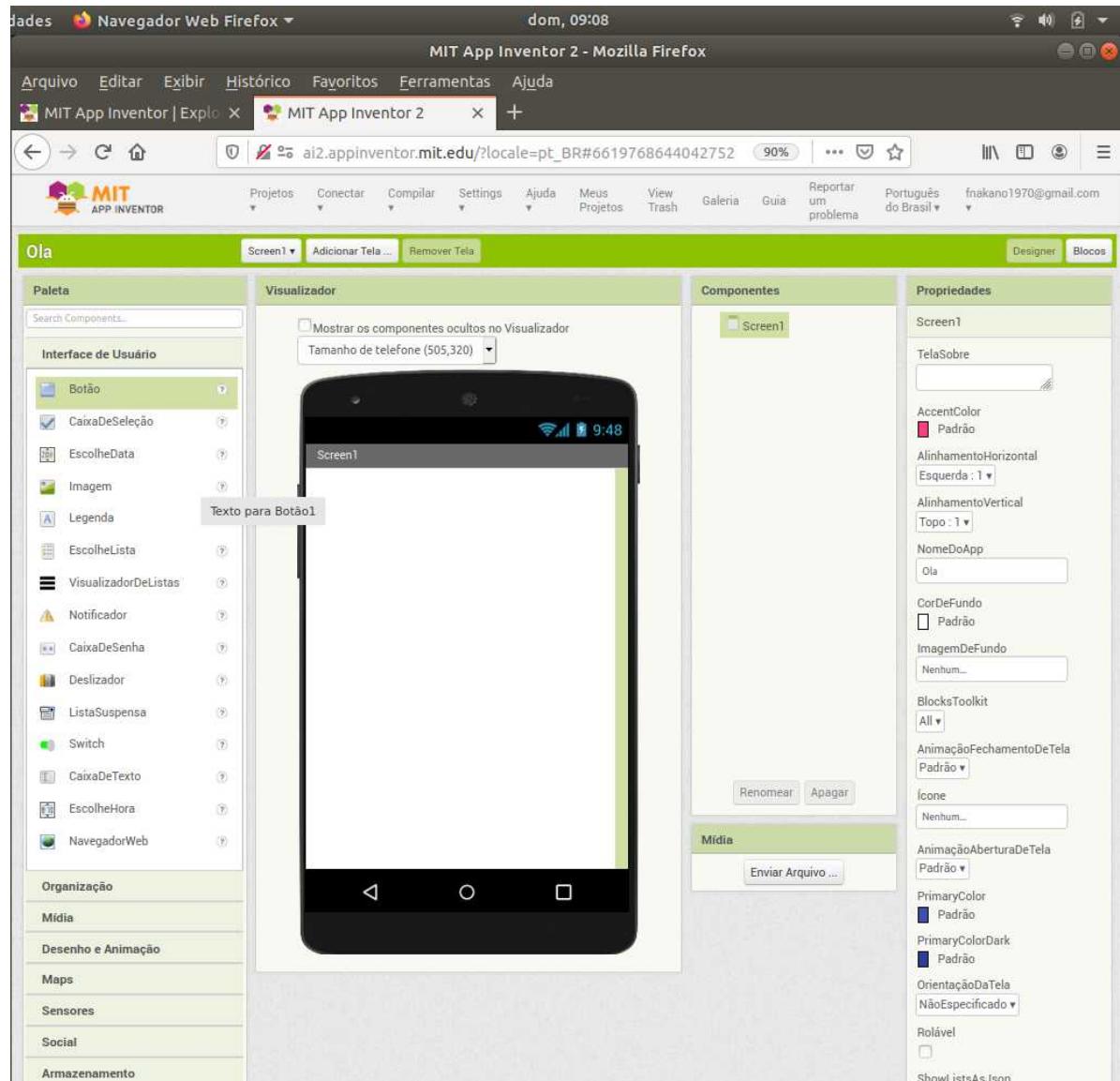
O quadro à esquerda abre na aba *Interface de Usuário*. Ela contém os componentes básicos dos aplicativos de celular.

Construir o primeiro App para celular

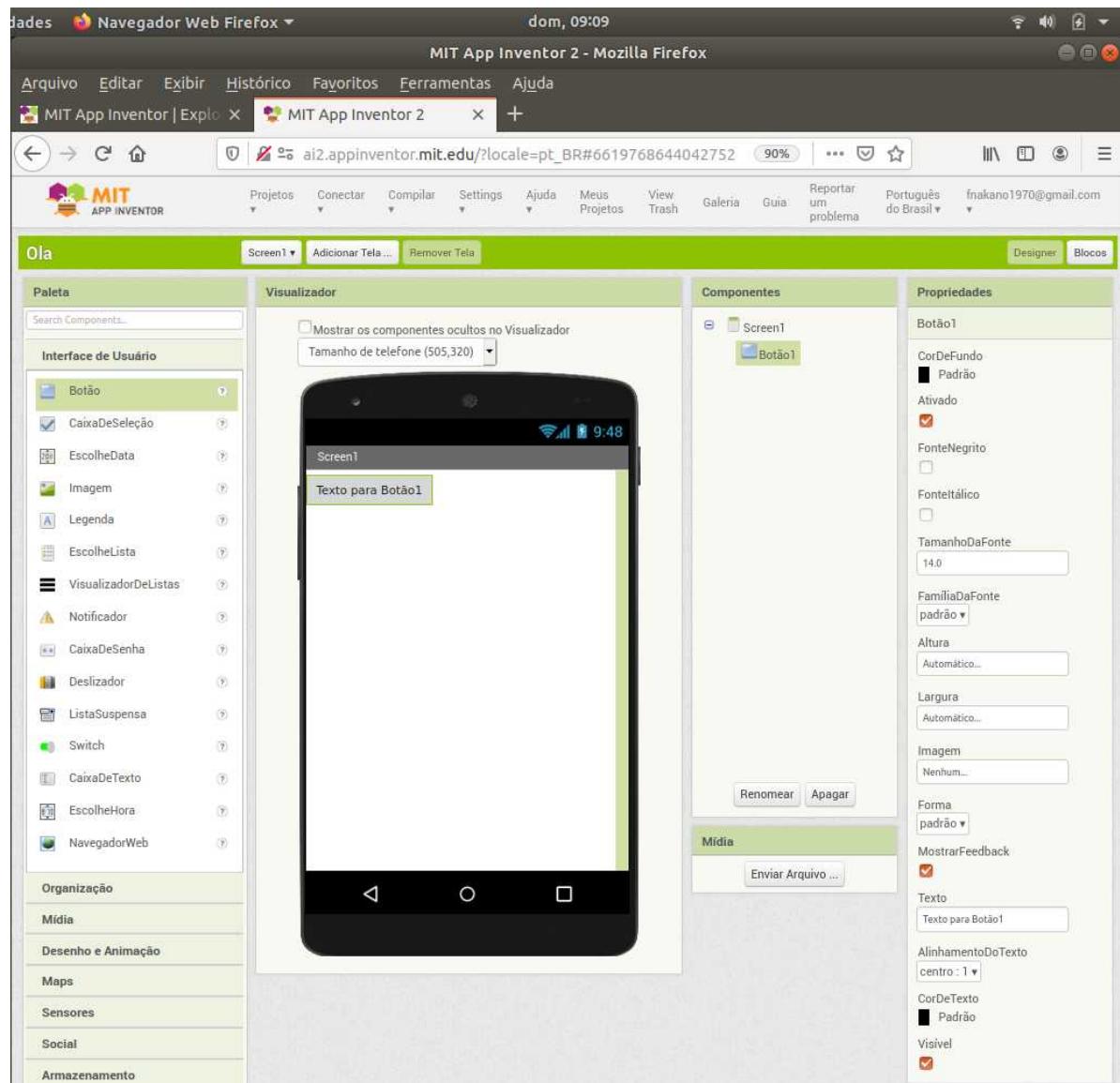
Vamos fazer um app em que digitamos nosso nome numa CaixaDeTexto, clicamos em um Botão e ele muda o texto em uma Legenda.

Começamos clicando e arrastando os componentes da lista da esquerda para a tela do celular...

Um Botão quando clicado, executa alguma ação.

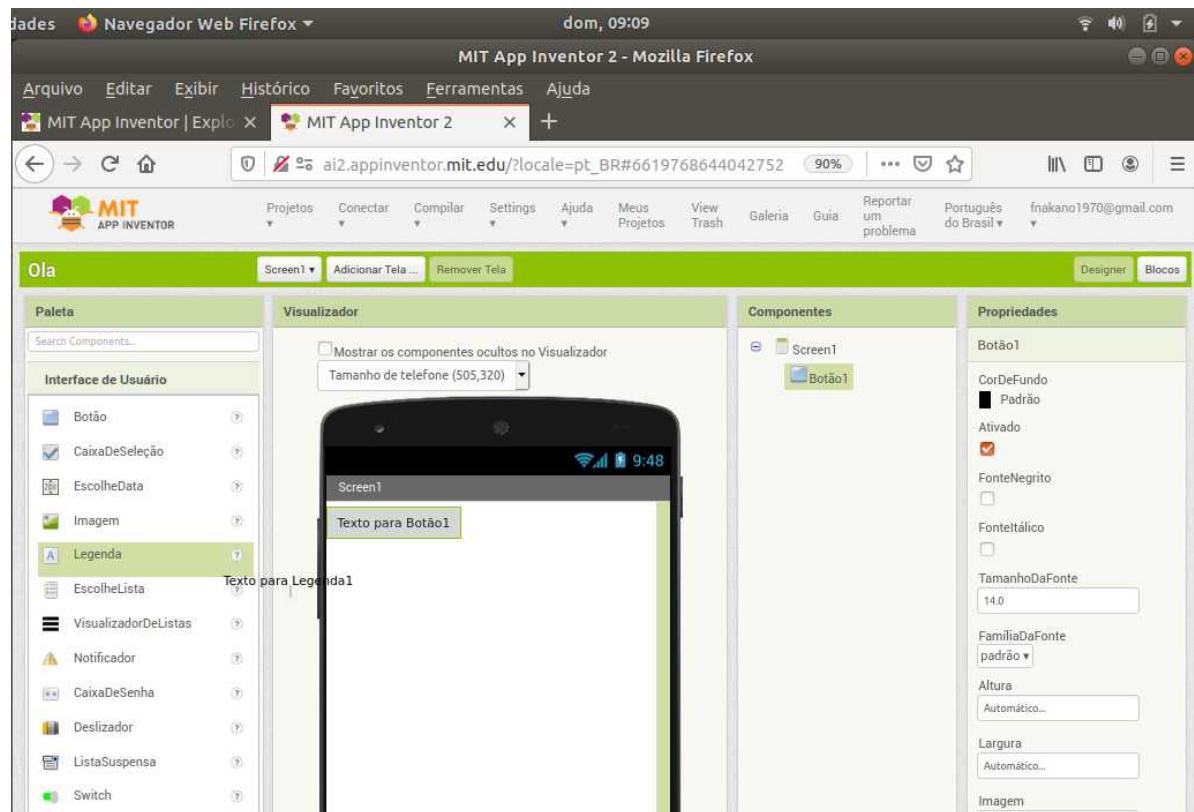


Construir o primeiro App para celular



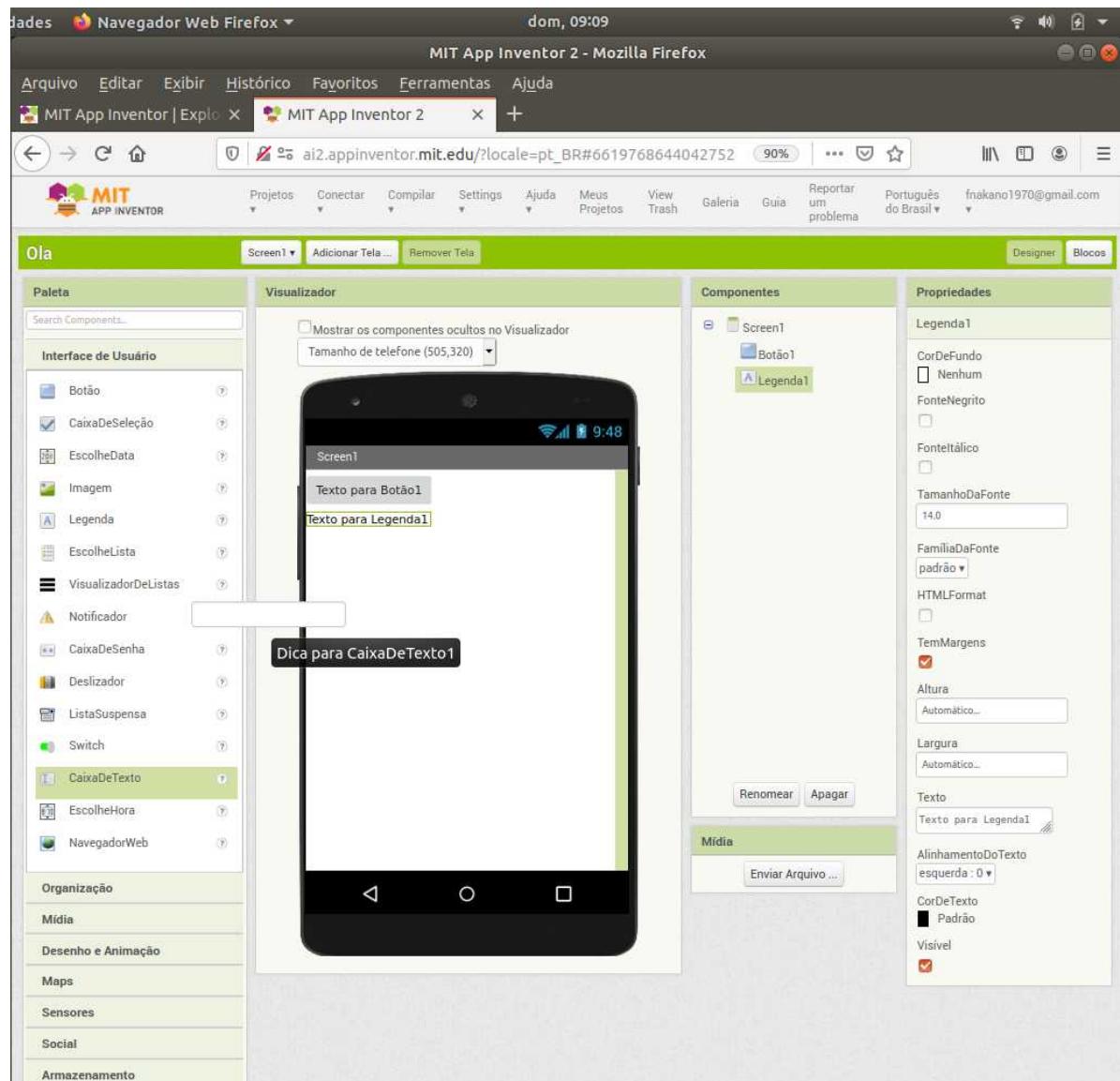
Uma Legenda é usada para mostrar texto.

Construir o primeiro App para celular

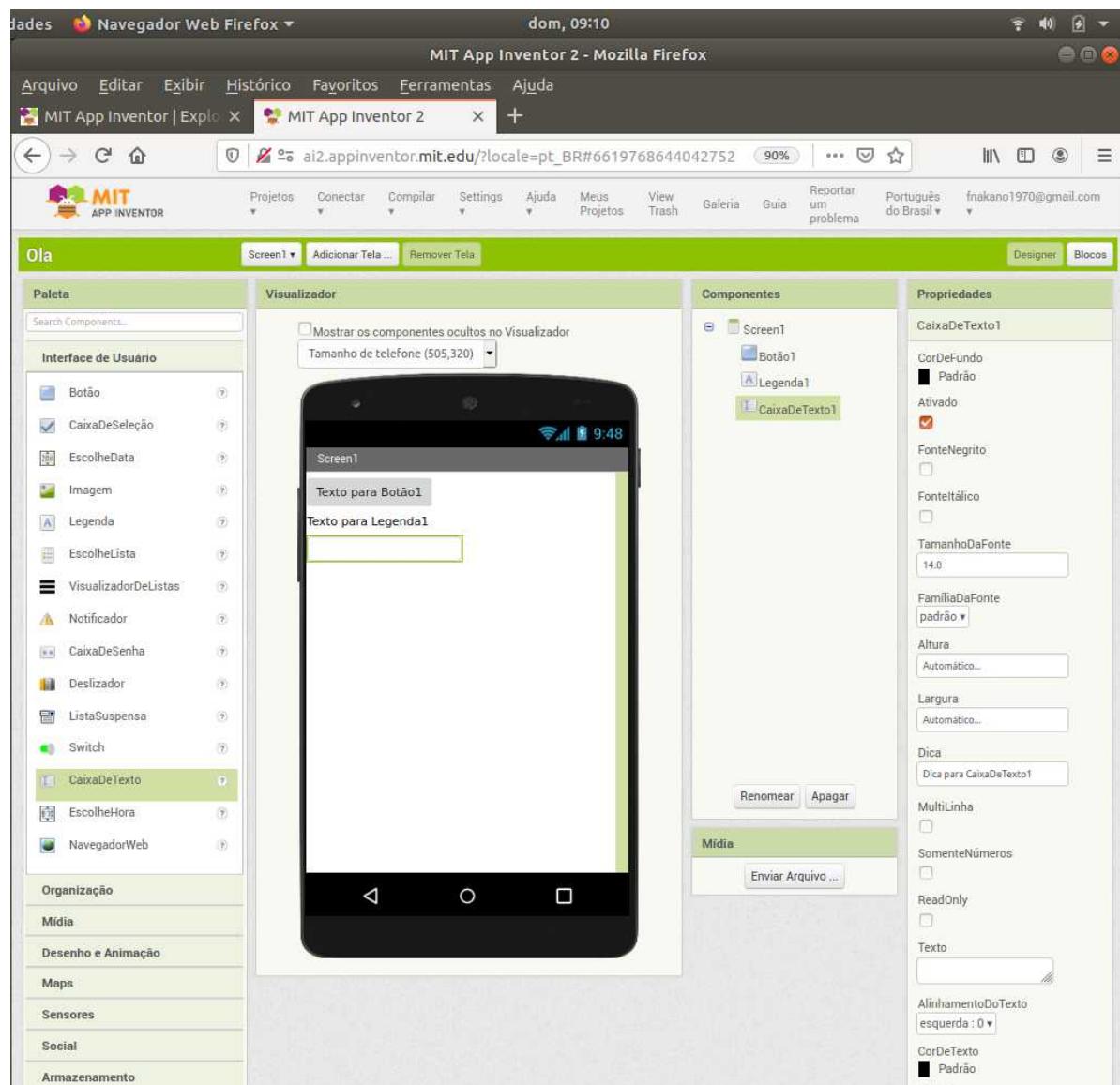


Uma CaixaDeTexto, quando clicada, abre o teclado e permite digitar texto dentro da caixa.

Construir o primeiro App para celular



Construir o primeiro App para celular



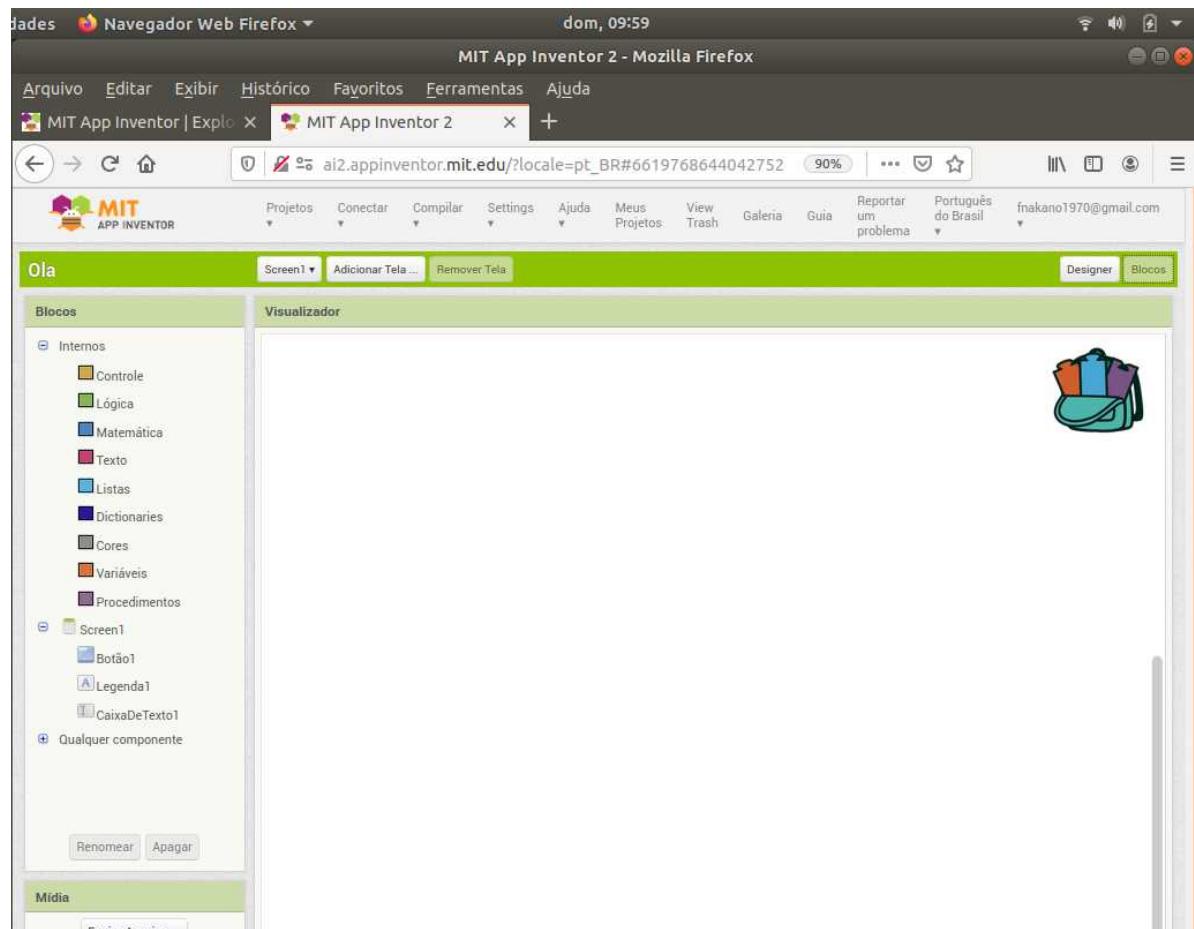
Terminamos o trabalho no modo designer. Neste modo nós escolhemos que componentes aparecem onde na tela do celular. As formas de dispor componentes na tela estão na aba Organização.

Definir as ações.

As ações do app são resultado do uso ordenado das ações dos componentes. Damos essa ordem dispondo blocos que representam as ações dos componentes. Para entrar no modo Blocos, no canto superior esquerdo da janela, clique no botão Blocos.

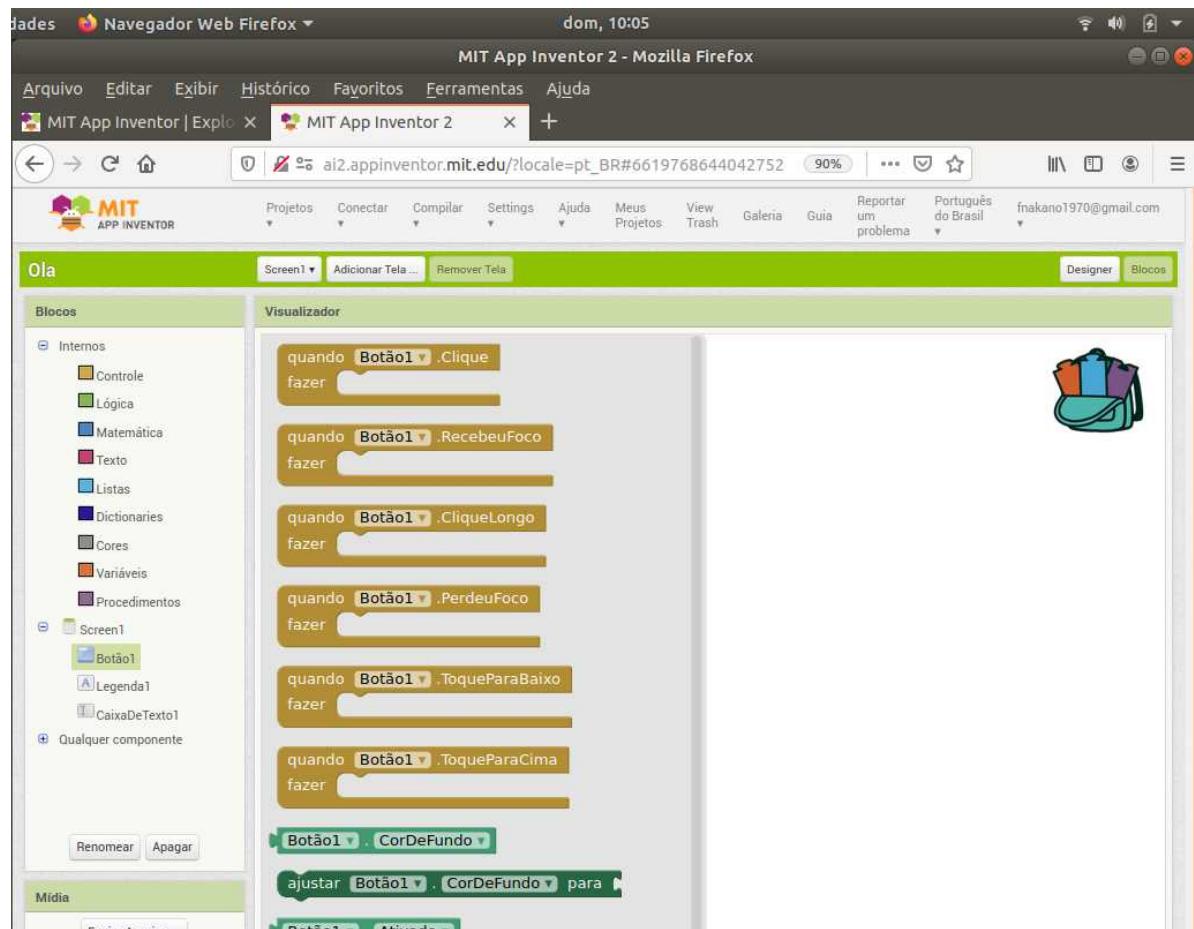
O modo blocos inicia com um plano vazio. À esquerda há um quadro com a aba Blocos aberta. Nesta aba são apresentados os componentes que podem ser usados. Há componentes internos, que estão disponíveis mesmo que não haja nada na tela do aplicativo e há os componentes que nós colocamos na tela no modo *design*. No caso, Screen1, Botão1, Legenda1 e CaixaDeTexto1.

Construir o primeiro App para celular



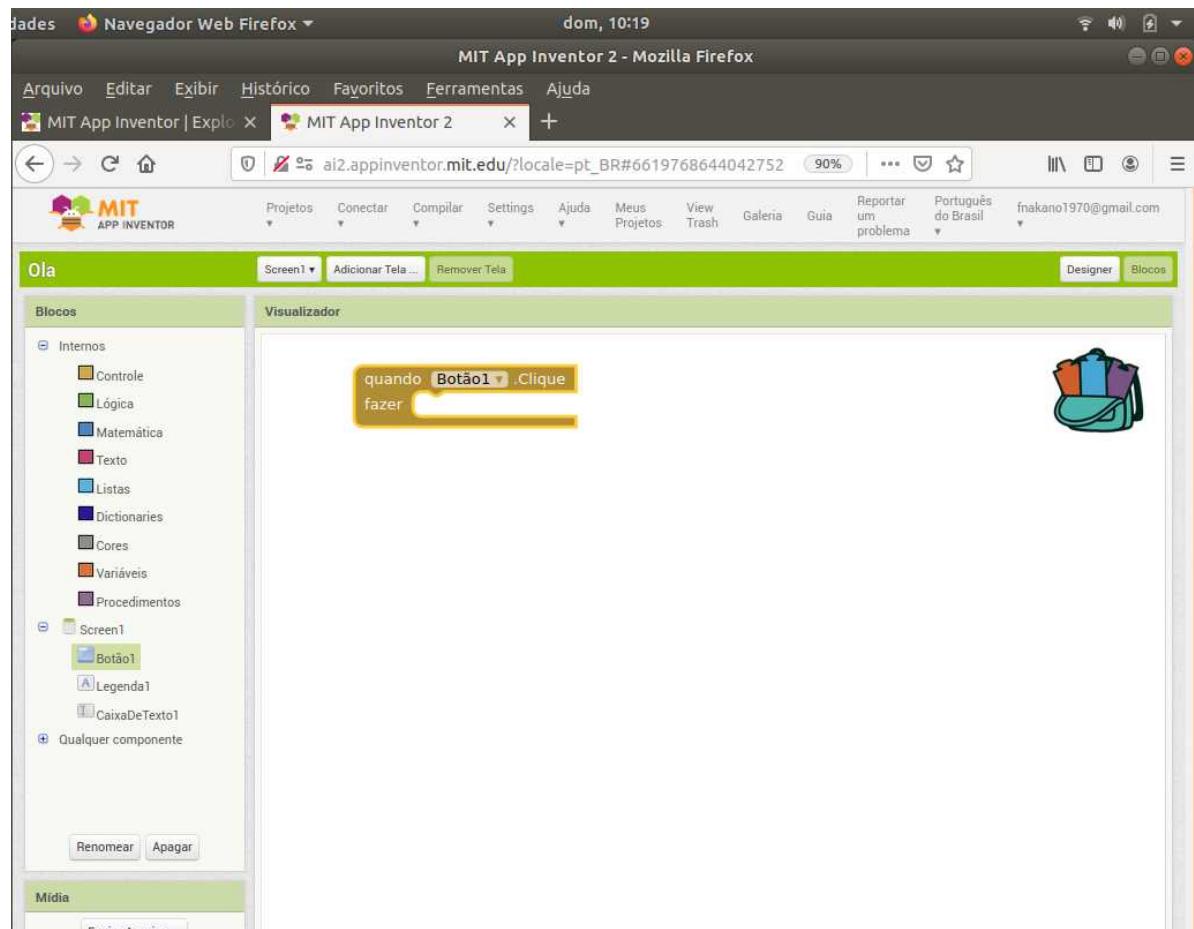
Clicando no componente abre-se a escolha de qual bloco do componente usar. O que é possível fazer com um componente no App Inventor está representado pelos blocos.

Construir o primeiro App para celular



Em um aplicativo, a execução dos blocos é iniciada por eventos determinados. Por exemplo, no Botão1, o bloco Clique (Botão1.Clique) é executado quando Botão1 for clicado durante a execução do app. Vamos arrastar o bloco Botão1.Clique para o plano.

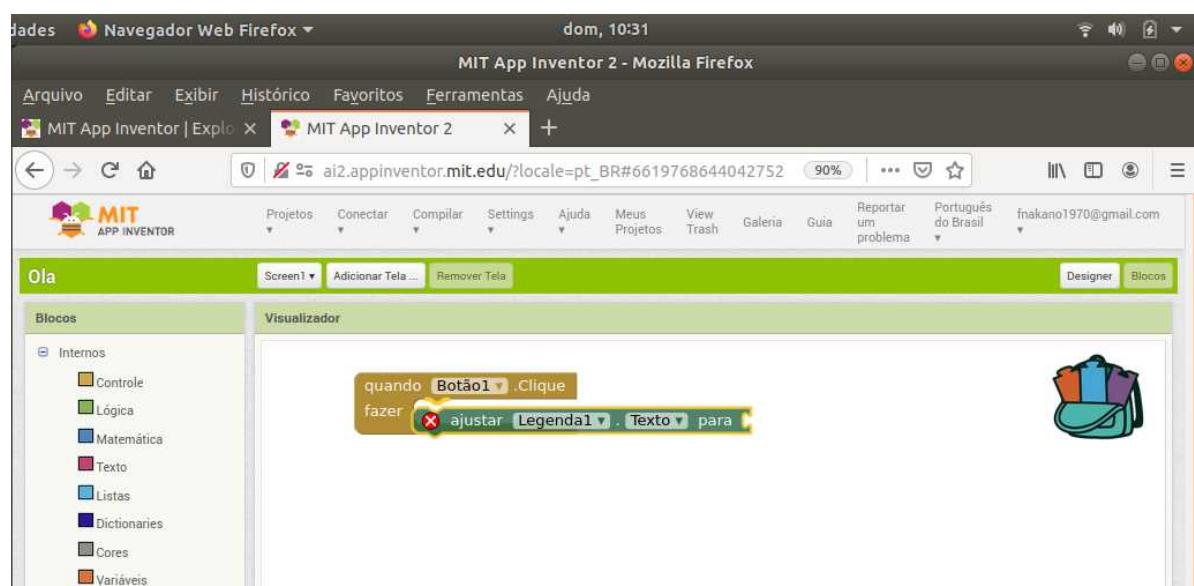
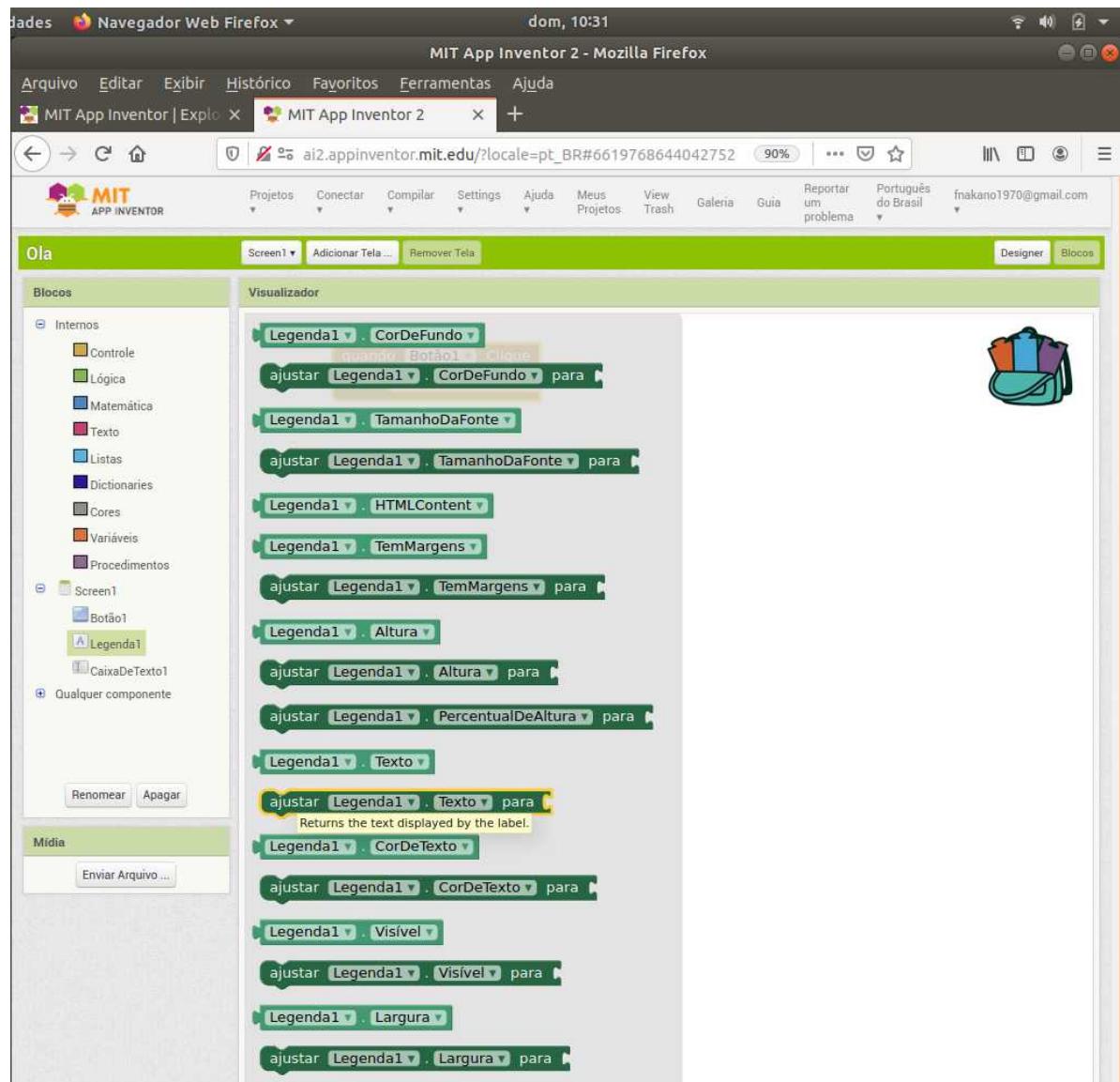
Construir o primeiro App para celular



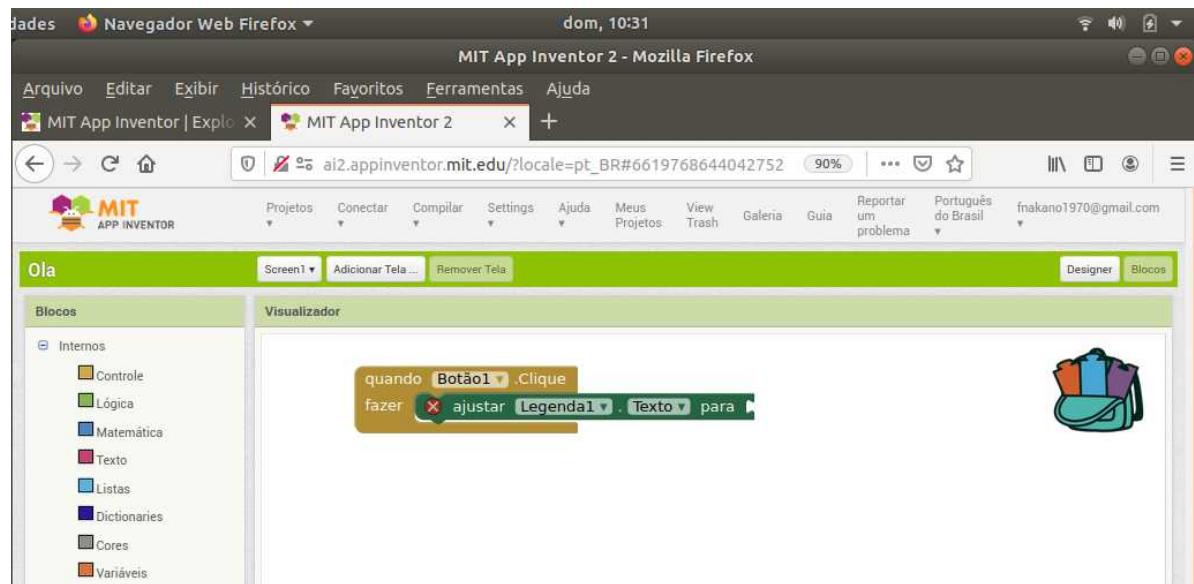
Queremos que o texto que digitarmos na CaixaDeTexto1 seja copiado para o texto da Legenda1. Para isso, vamos ajustar Legenda1.texto para CaixaDeTexto1.texto.

Clique em Legenda1 e arraste o bloco ajustar Legenda1.texto e encaixe dentro do Botão1.Clique.

Construir o primeiro App para celular

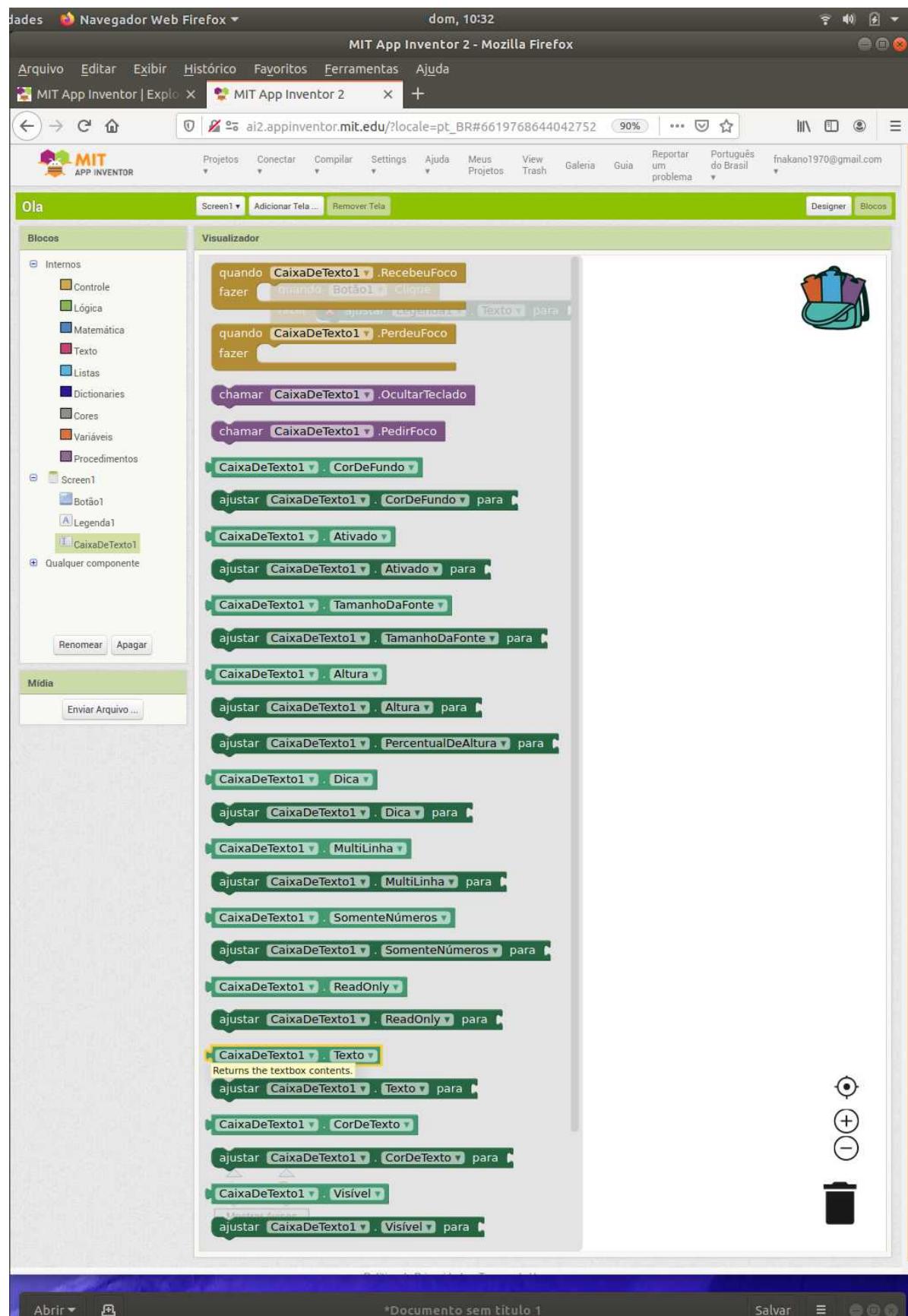


Construir o primeiro App para celular

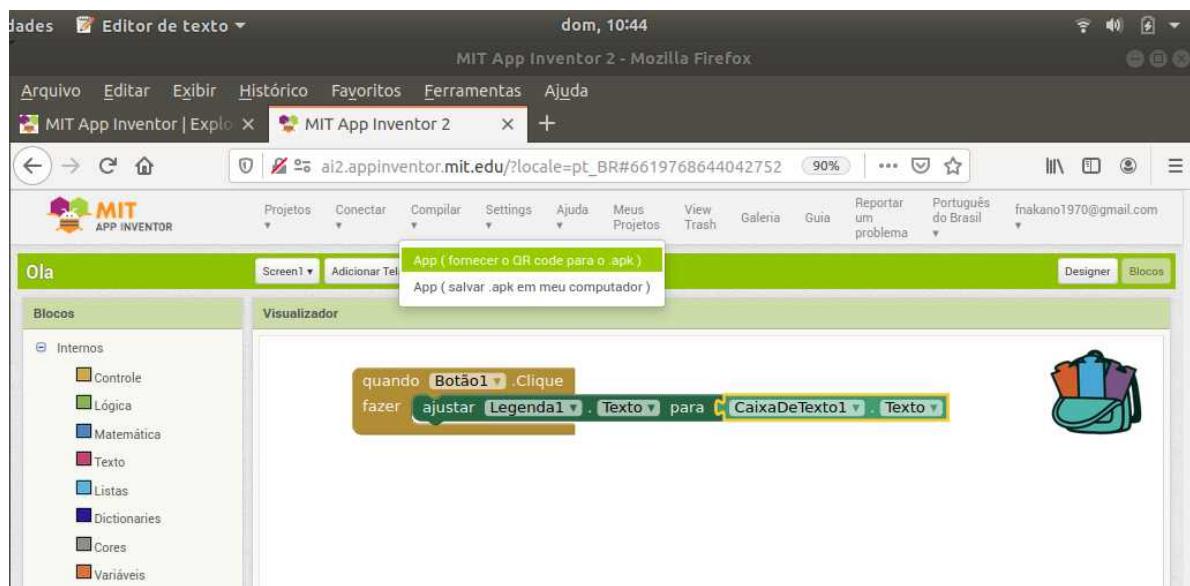


Clique em CaixaDeTexto1, arraste o bloco CaixaDeTexto1.Texto e encaixe em ajustar Legenda1.Texto.

Construir o primeiro App para celular



Construir o primeiro App para celular



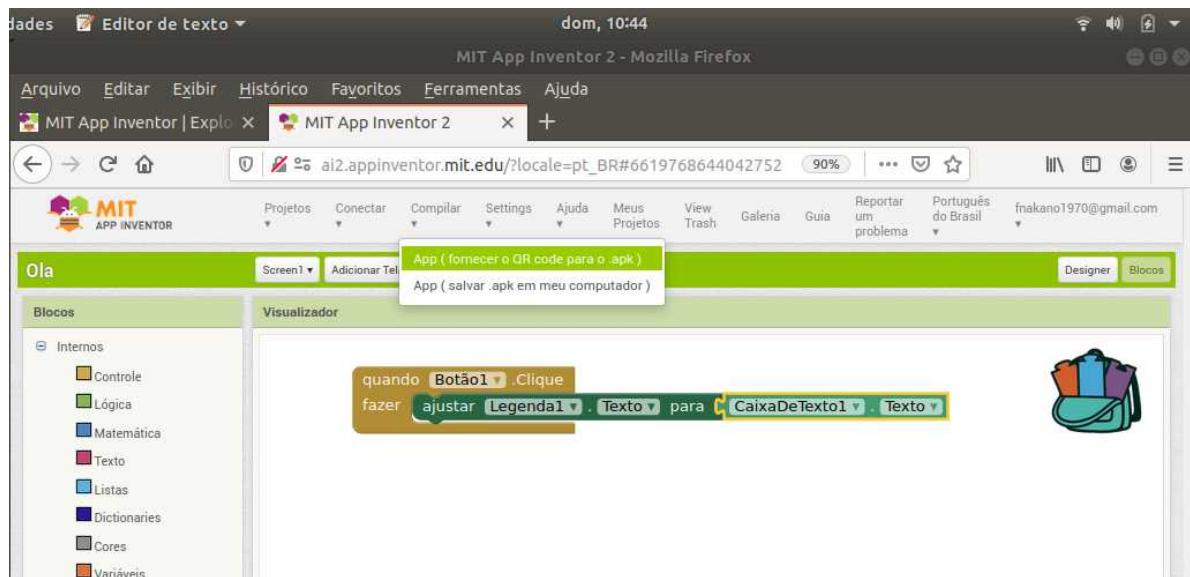
Vamos reler os blocos para checar o conjunto:

Quando Botão1.Clique fazer Ajustar Legenda1.Texto para CaixaDeTexto1.Texto

Nesta forma de programar, isto é suficiente para quando o botão for clicado, copiar o texto da caixa de texto para a legenda.

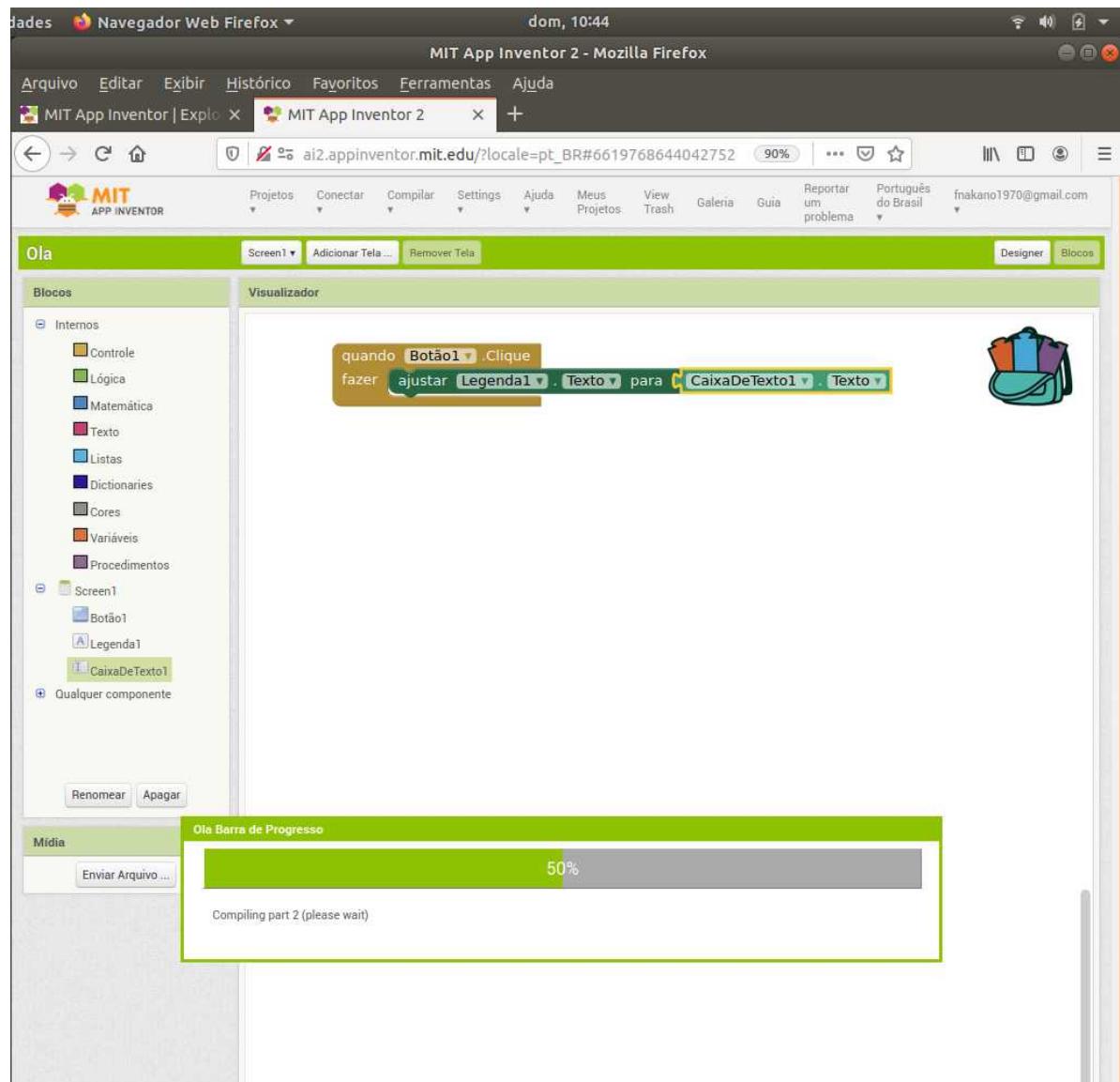
Enviar o app para o celular.

Para gerar o app de celular, na barra de menu do alto da janela selecione Compilar -> App (fornecer o QR code para o .apk).



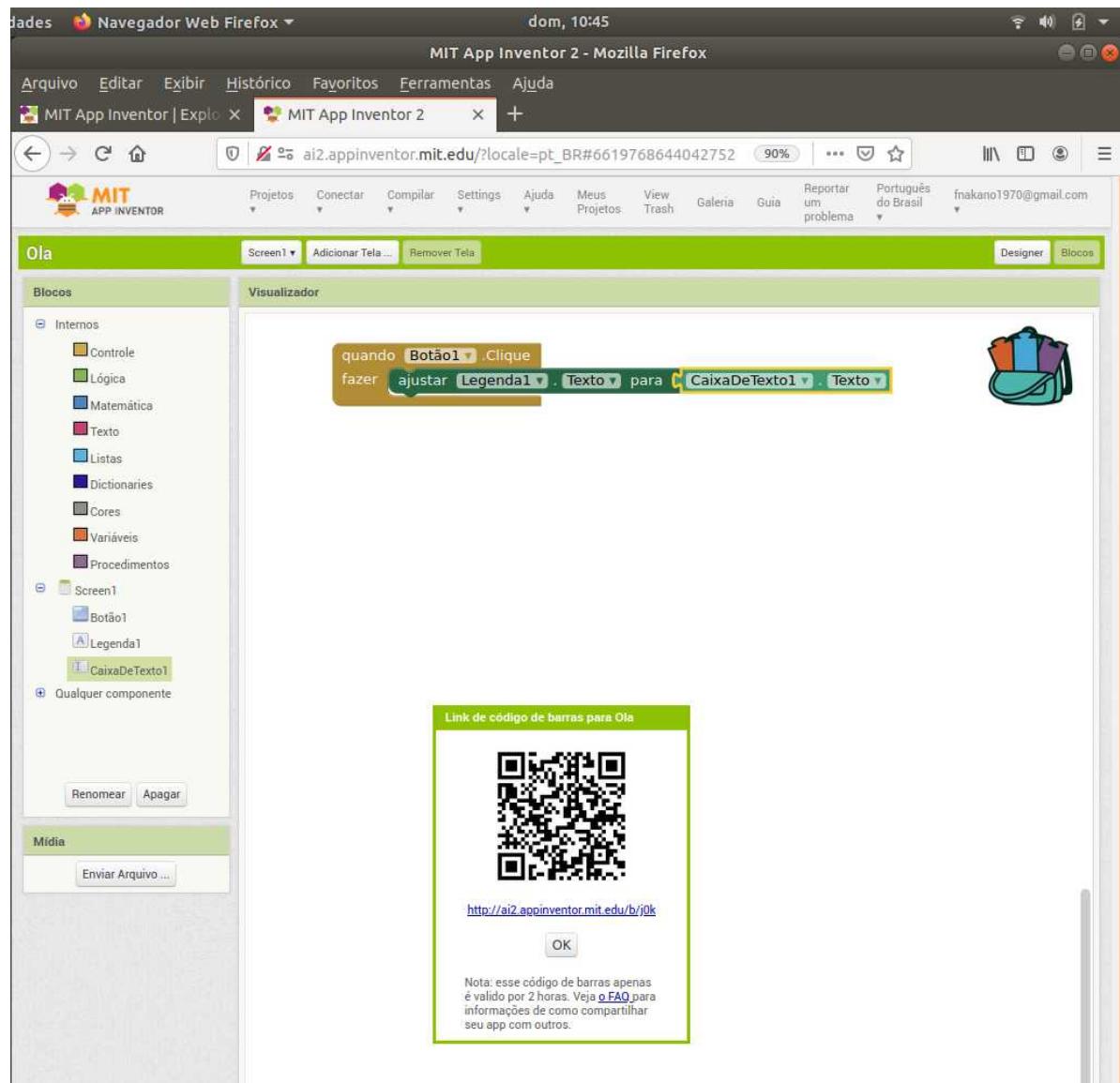
Compilação é o nome dado em computação para o processo de traduzir de uma linguagem para outra. No caso, compila-se da linguagem de programação em blocos para a linguagem dos aplicativos de celular. O resultado da compilação, neste caso, é um aplicativo de celular.

Construir o primeiro App para celular



Além de gerar o aplicativo, o App Inventor fornece um link para baixar o aplicativo. Este é passado para nós por um QR-Code.

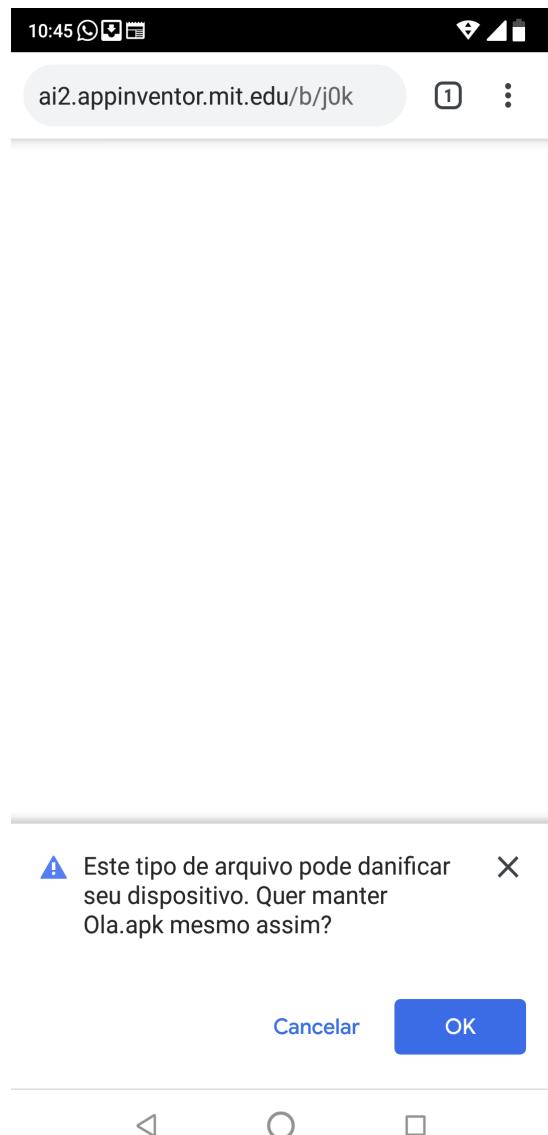
Construir o primeiro App para celular



Testar o app

Abra o leitor de QR-Code, leia o código e clique no link. Um programa pode transmitir os dados do celular para alguém, ou, usar o celular para enviar mensagens para os contatos armazenados nele, ou até mesmo danificar o celular. Por isso quem está baixando o programa precisa ter certeza do que está fazendo. Para certificar, aparecerá uma tela para confirmar que você deseja baixar o arquivo que contém seu programa.

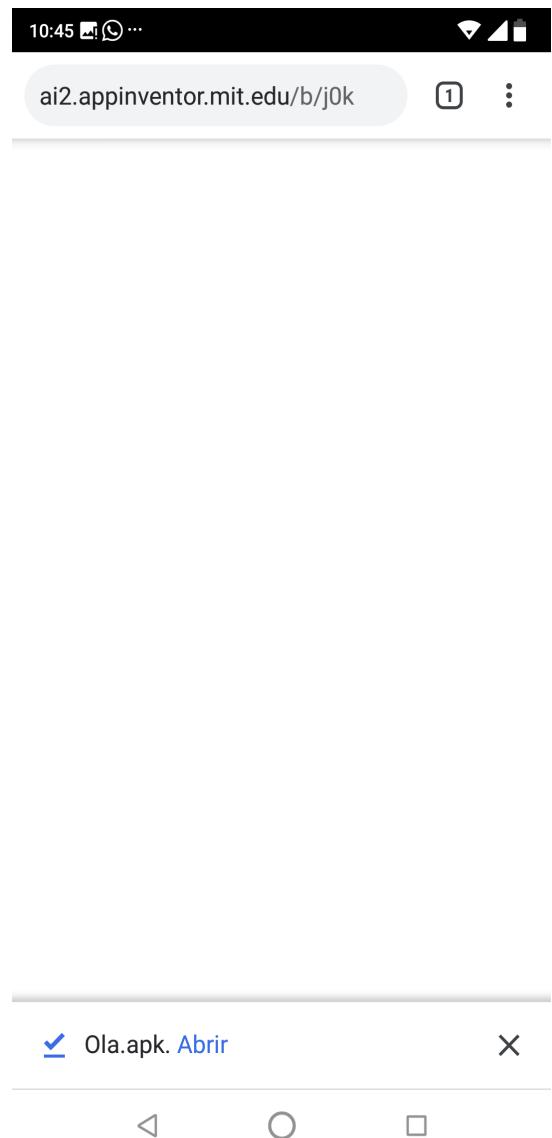
Como é o programa que você fez, usando sites e instruções confiáveis, deve ser seguro baixá-lo. Se você concorda, responda OK.



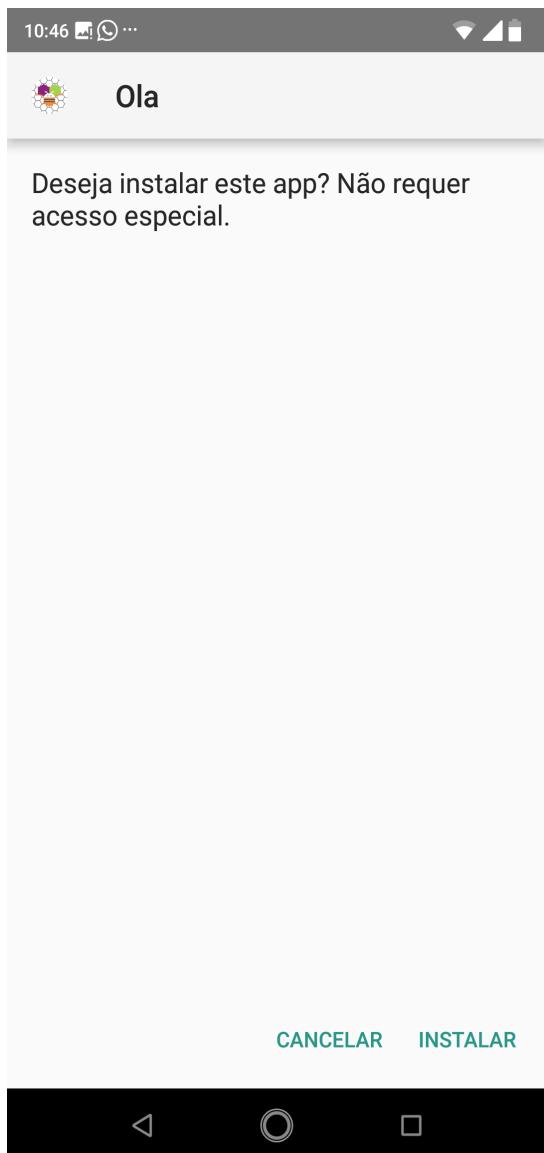
(**nota:** imagens são tratadas como caracteres.)

Depois abra o programa (clique em abrir). Se a notificação sair da tela, entre no navegador em downloads, encontre Ola.apk e dê um duplo-clique sobre ele (isto equivale a clicar em abrir).

Construir o primeiro App para celular



Em seu celular, na tela a seguir, clique em Instalar



Aguarde o processo de instalação terminar

Construir o primeiro App para celular



! Sistema Android ▾

Não foi possível obter a captura de tela.

Não é possível salvar a captura de tela, porque não ..



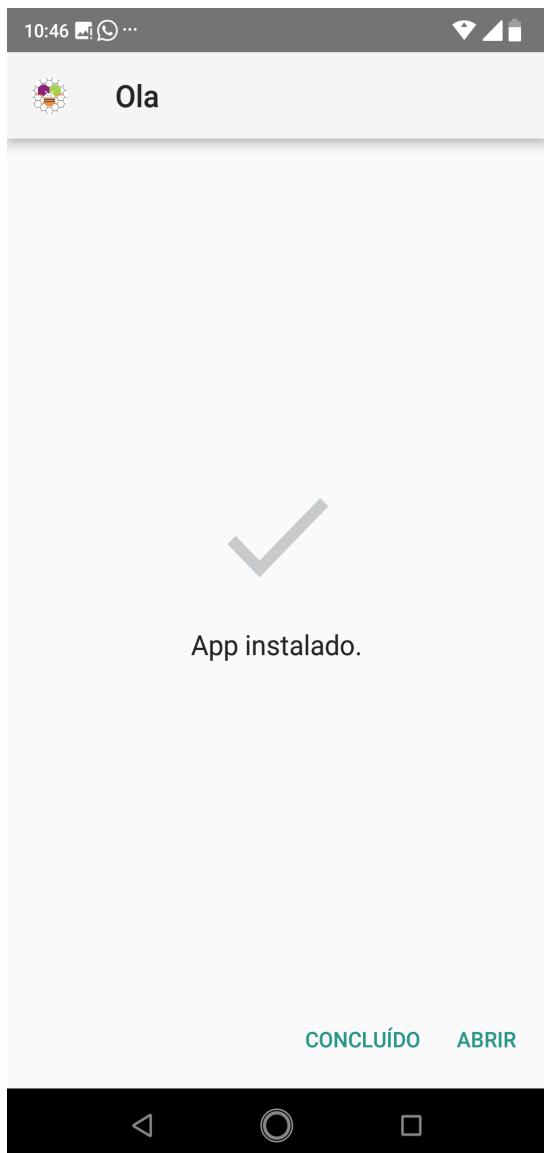
Instalando...

CANCELAR



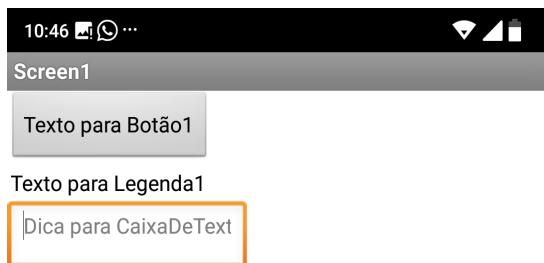
Terminada a instalação clique em abrir

Construir o primeiro App para celular



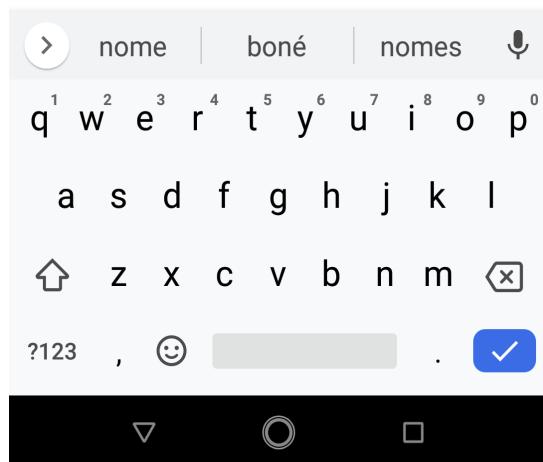
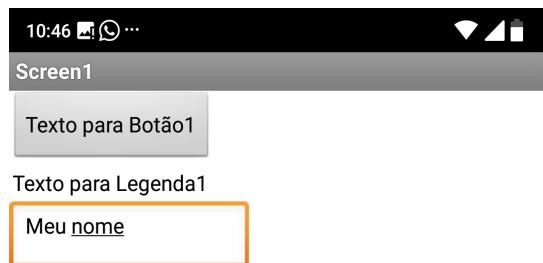
Você verá a tela que você criou

Construir o primeiro App para celular



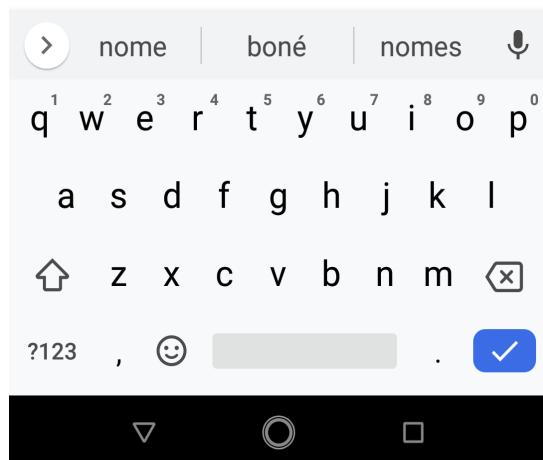
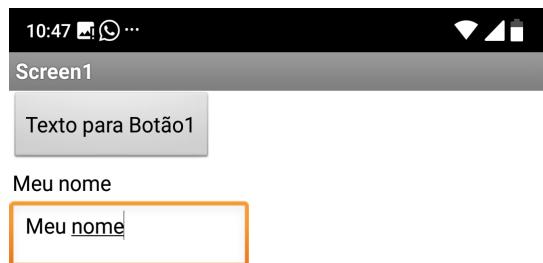
Clique na caixa de texto - o teclado aparecerá, digite o texto que quiser e clique no botão,

Construir o primeiro App para celular



o texto que você digitou aparecerá na Legenda1.

Construir o primeiro App para celular



Parabéns, você completou todos os passos para construir um app Android.

Aplicativos para teste do ImageCV

Aplicativo para binarizar cores a partir de foto de arquivo (Ola.apk) Aplicativo para localizar elipses (Elipses.apk) Aplicativo para localizar quadriláteros (Retangulos.apk)

Código-fonte do Aplicativo para binarizar cores a partir de foto de arquivo (Ola.aia) Código-fonte do Aplicativo para localizar elipses (Elipses.aia) Código-fonte do Aplicativo para localizar quadriláteros (Retangulos.aia)