B.Up
EDUCACIONAL
SOLUÇÕES PEDAGÓGICAS

PROGRAMAÇÃO
E DESENVOLVIMENTO DE APLICATIVOS COM APP INVENTOR

1 APP INVENTOR

MIT
APP INVENTOR

Programação e Desenvolvimento de Aplicativos com App Inventor
2022

INDICE

INTRODUÇÃO

Umberto Eco, um grande escritor italiano do século XX, um dia disse que "o computador não é uma máquina inteligente que ajuda pessoas burras; ao contrário é uma máquina burra que só funciona nas mãos de pessoas inteligentes". E o jeito mais inteligente de fazer funcionar esta "máquina burra" é programando-a, ou seja, dizendo a ela o que ela deve fazer.

É isso o que um programador de computador faz: ensina o computador a pensar. E o mais interessante é que, ao fazer isso, ele, o programador, acaba aprendendo um pouco mais sobre como que ele próprio pensa. É por isso que muitos estudiosos do pensamento humano acreditam que programar computadores seja um ótimo exercício intelectual, quer dizer, pessoas inteligentes que ajudam as máquinas burras a funcionarem de forma inteligente, acabam ficando ainda mais inteligentes no processo.

Pessoas inteligentes se tornam ainda mais inteligentes quando aprendem a programar!

Programação de Computadores é a arte de ensinar o computador. Através da Programação de Computadores ampliamos as formas de nos comunicarmos e nos relacionarmos com os aparatos tecnológicos. Deixamos de ser apenas usuários de tecnologia e passamos a ser produtores de novos recursos, como jogos e aplicativos, por exemplo.

Programar é expressar, na forma de um conjunto de instruções, o que a máquina irá executar. Programar é criar.

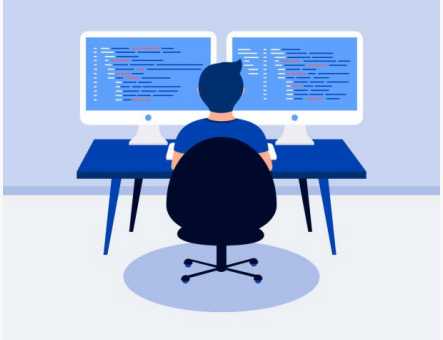


Figura 1: ilustração 'programador de computadores 20.662' (iStock)

Mas como fazer isso? Como ensinar uma máquina a pensar? O primeiro passo é conseguir se comunicar com ela, ou seja, aprender a falar a sua língua e, para isso, você vai precisar de uma "Linguagem de Programação". A linguagem de programação é o elo de ligação entre a pessoa (inteligente) e o computador (burro). E existem muitas! Elas são tão antigas quanto os computadores.

Um computador é uma máquina ultra inteligente, mas ele precisa receber todas as instruções necessárias para que algo seja feito. E nós, programadores, somos os responsáveis por enviar essas instruções através das linguagens de programação.

Para os jovens, aprender a programar traz uma série de benefícios importantes para melhorar o desempenho escolar, como exercício do raciocínio lógico, foco, concentração, análise e resolução de problemas, trabalho em equipe, senso de organização, e muitos mais.

Figura 2: criança programadora (Getty Images, iStock)

Ao atingirem a vida adulta, esses jovens terão habilidades e competências diferenciadas, que além de torná-los profissionais mais capacitados, os colocará em posição de destaque no mercado de trabalho.

INTRODUÇÃO

Aplicativos para dispositivos móveis são uma importante peça de nosso relacionamento com a tecnologia. Desde 2008, quando os primeiros *smartphones* foram lançados, os aplicativos deram aos proprietários desses aparelhos o poder de decidir que tipo de funcionalidades deveriam estar presentes para seu conforto. Isso trouxe para os usuários o poder de escolha e personalização de suas ferramentas.

Além disso, com as lojas de aplicativos, qualquer programador hoje é capaz de criar seus app e disponibilizá-lo para todos os usuários de uma determinada classe de *smartphones* e *tablets*. Ou seja, a solução encontrada pelos programadores é compartilhada de maneira quase instantânea, e cada usuário julga se aquela solução é ou não útil para seu uso. Isso faz com que cada aparelho seja único, com uma combinação de aplicativos que satisfaça apenas seu dono.

Tendo isso em vista, a criação de aplicativos para dispositivos móveis vem se tornando um grande mercado para os desenvolvedores de software, com diversas oportunidades surgindo. Por isso, decidimos trazer para os alunos esse conteúdo, baseado na ferramenta App Inventor do MIT, para auxiliar os alunos que queiram iniciar suas jornadas pelo universo de desenvolvimento de aplicativos móveis.

"A criação de aplicativos para dispositivos móveis vem se tornando um grande mercado para os desenvolvedores de software, com diversas oportunidades surgindo"

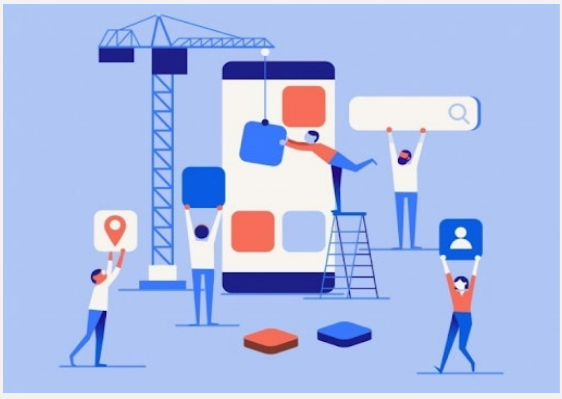


Figura 3: desenvolvimento de aplicativos

O App Inventor é uma ferramenta criada pelo MIT (Massachusetts Institute of Technology) para auxiliar no desenvolvimento de aplicativos para a plataforma Android. Lançado inicialmente em dezembro de 2010, o App Inventor traz diversas facilidades, dentre as quais a que mais se destaca é o uso de uma linguagem de programação *drag and drop* semelhante ao Scratch, em que o programador deverá arrastar os blocos de comandos para realizar sua programação.

Recentemente o MIT lançou a versão 2 do App Inventor, com novas funcionalidades e uma interface mais amigável e limpa. É essa a versão que usaremos no decorrer deste curso.

CONTEÚDO

Este módulo inicial tem como macro objetivo introduzir o estudo do APP INVENTOR aos alunos, apresentando a eles as principais ferramentas e funcionalidades do programa.

Durante as aulas serão desenvolvidos projetos que, além de ensinarem a programar aplicativos, irão trabalhar também conceitos importantes (ainda que introdutórios neste primeiro momento) de matemática, como condicionais, variáveis e coordenadas cartesianas, passando por uma breve introdução a banco de dados e geolocalização.

Este curso pode ser considerado "pré-profissionalizante", pois já proporcionará ao aluno conhecimento para começar a se aventurar pelo promissor mercado de desenvolvimento.

Neste primeiro momento, o conteúdo será apresentado parcialmente em português e parcialmente em inglês, para facilitar a compreensão. Nos módulos seguintes, todo o ambiente App inventor será apresentado em Inglês.

A seguir, veremos os conteúdos e objetivos detalhados de cada unidade deste material, para facilitar sua compreensão.

CONTEÚDO

UNIDADE 1	**Projeto 'S.O.S'**	OBJETIVOS • Conhecer o ambiente App Inventor e os componentes básicos da ferramenta • Programação – modelagem IPO
UNIDADE 2	**Projeto 'Desenho Livre'**	OBJETIVOS • Avançar em modelagem IPO de programação • Introdução a componentes de Layout e Sensores • Estudo de coordenadas cartesianas
UNIDADE 3	**Projeto 'Desenho Livre 2.0'**	OBJETIVOS • Adição de 'dinâmica' aos aplicativos • Introdução à mudança e transição de telas • Trabalhar com propriedades dos elementos
UNIDADE 4	**Projeto 'Calculadora IMC'**	OBJETIVOS • Trabalhar o processamento do aplicativo • Aprofundamento em aspectos de programação • Introdução às "Instruções Condicionais"
UNIDADE 5	**Projeto 'Teste de Reflexos'**	OBJETIVOS • Avançar para aplicativos dinâmicos • Introdução ao componente Temporizador • Introdução ao componente SpriteImagem
UNIDADE 6	**Projeto 'Mapa Turístico'**	OBJETIVOS • Entender como os aplicativos se conectam a bases de dados externas • Conexão ao Google Maps • Introdução ao conceito de geolocalização
UNIDADE 7	**Projeto 'Game Pong'**	OBJETIVOS • Entender a mecânica dos games • Trabalhar com colisões de Sprites • Reforço de conceitos de Álgebra e Geometria
UNIDADE 8	**Projeto 'Mini Golf'**	OBJETIVOS • Utilizar novos gestos para Sprites • Reforço de conceitos de Álgebra e Geometria • Avançar na programação de aplicativos
UNIDADE 9	**Projeto 'Fusion Tables'**	OBJETIVOS • Aprender a utilizar sensor de localização • Introdução ao conceito de banco de dados • Reforço dos conceitos de geolocalização
UNIDADE 10	**Projeto 'Quiz Ilustrado'**	OBJETIVOS • Aprender a programar múltiplos fatores • Exercitar conhecimentos gerais

CONTEÚDO

UNIDADE 11	**Projeto 'Quiz Ilustrado II''**	OBJETIVOS • Aprofundar-se na programação de múltiplos fatores • Praticar o conceito de 'melhoria contínua'
UNIDADE 12	**Projeto 'Lista de Compras'**	OBJETIVOS • Introdução ao componente *ListPicker* • Programar inserção e exclusão de itens
UNIDADE 13	**Projeto 'Lista de Compras II'**	OBJETIVOS • Adição de 'dinâmica' aos aplicativos • Introdução à mudança e transição de telas • Trabalhar com propriedades dos elementos
UNIDADE 14	**Projeto 'Quiz Múltipla Escolha'**	OBJETIVOS • Trabalhar o processamento do aplicativo • Aprofundamento em aspectos de programação • Introdução às "Instruções Condicionais"
UNIDADE 15	**Projeto final - o grande desafio!**	OBJETIVOS • Projeto de avaliação final

PRIMEIROS PASSOS

Acesso ao ambiente App Inventor

O App Inventor é uma ferramenta online, ou seja, é necessária uma conexão com a internet para que ele seja utilizado. Além disso, um navegador recente é necessário para que você consiga acessar a página em que a ferramenta está armazenada.

Acessando o endereço http://ai2.appinventor.mit.edu/ você verá uma tela como a seguinte:

Figura 4: 1ª acesso ao App Inventor

Caso você já possua uma conta do Google (normalmente associada a serviços Google como Gmail e Youtube), coloque seu nome de usuário no campo 1. Caso não possua uma conta Google e deseje criar uma, clique no link 2. Possuir uma conta Google é necessária para que você acesse o site do App Inventor e seus projetos em sempre que desejar. Se não quiser seu e-mail pessoal exposto em projetos de aplicativos, recomendamos que crie uma nova conta apenas para desenvolvimento deles.

Quando você entrar pela primeira vez em sua conta no App Inventor, verá a tela a seguir:

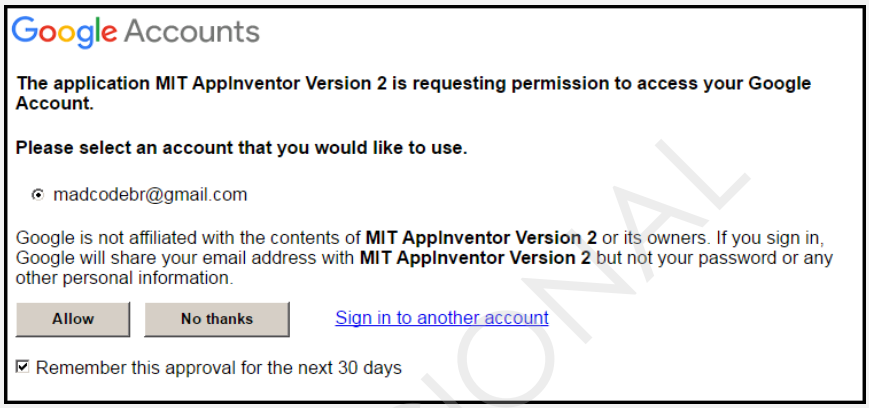


Figura 5: 1ª acesso ao App Inventor

Clique em *Allow* para permitir que o App Inventor interaja com sua conta Google. Isso permitirá a sincronização de sua conta com seus projetos, além de permitir que aplicativos desenvolvidos que tenham a tecnologia de envio de mensagens o façam partindo de seu e-mail. Uma vez dentro do sistema, você verá uma tela como a que está a seguir.

Ao longo deste material, conheceremos todos os menus presentes nessa tela. Para iniciarmos nosso primeiro projeto, clique ao lado de onde está seu e-mail, para selecionar a língua em que iremos programar. Para este primeiro módulo, selecione Português do Brasil. Nos próximos, trabalharemos em Inglês.

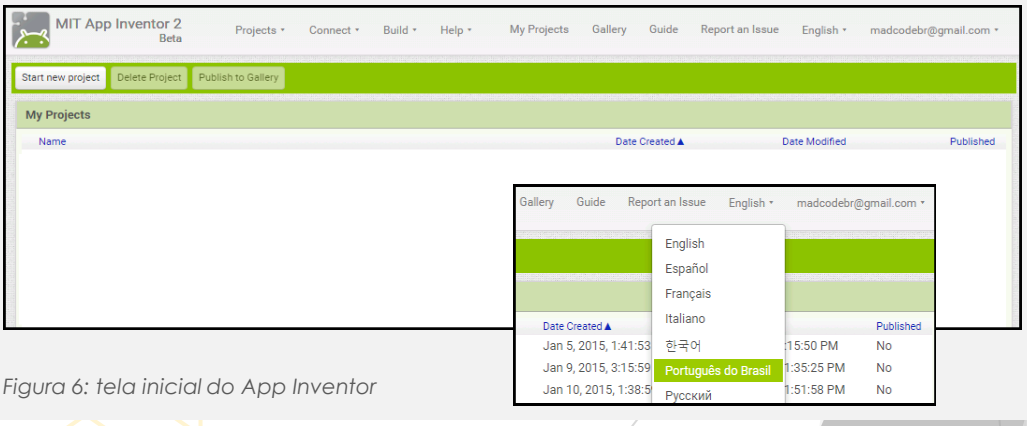


Figura 6: tela inicial do App Inventor

Ambiente *Designer*

Ao iniciar um novo projeto, a primeira tela que aparecerá será a que você pode ver a seguir. Nela estão todos os elementos necessários para a criação de um aplicativo.

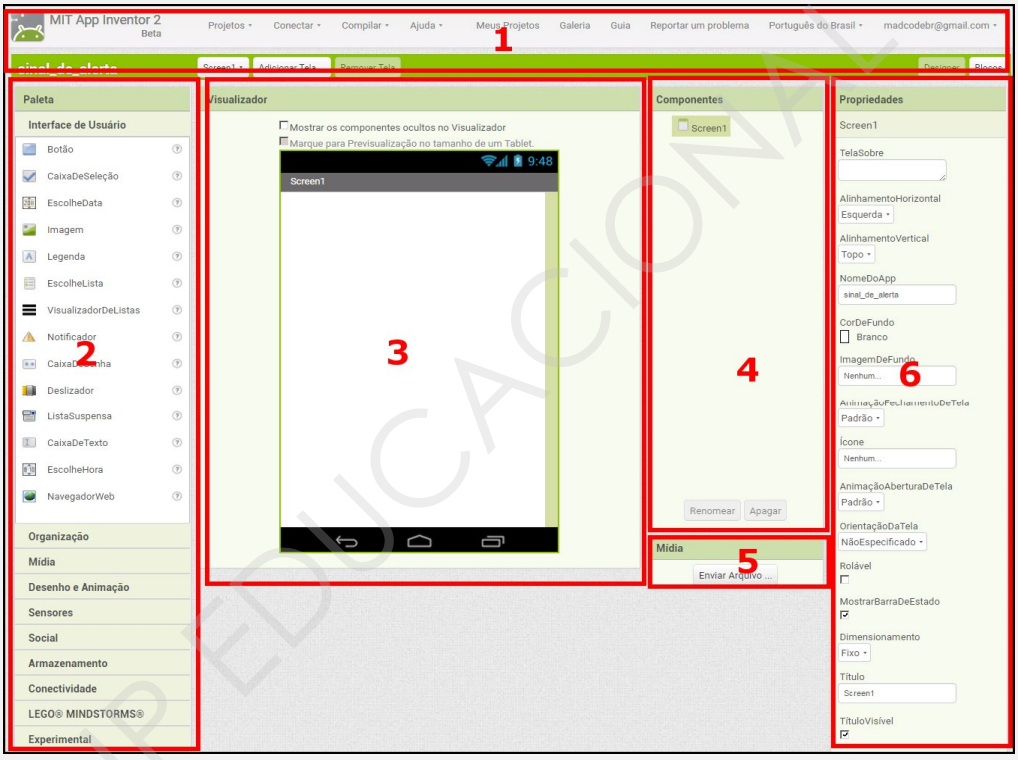


Figura 7: ambiente Designer

No ambiente Designer, você irá construir a aparência de seu aplicativo, além de adicionar os componentes que serão necessários para o seu desenvolvimento.

A tela do Designer é dividida como você pode ver na imagem.

1. **Menu:** Contém comandos para manipular os arquivos dos aplicativos feitos, para compila-los, para construir os arquivos *.apk* (necessários para instalação nos aparelhos Android). Você já acessou essa região do Designer quando teve que alterar o idioma.
2. **Paleta:** é uma lista de componentes a partir dos quais pode escolher. A paleta é dividida em seções, das quais apenas os itens da seção *Interface de Usuário* são visíveis quando você abre o programa. Você pode clicar nas outras seções (*Organização*, *Mídia*, *Sensores*, etc) para ver os componentes de cada uma delas.
3. **Visualizador:** Nessa área você irá organizar os componentes de seu aplicativo da maneira como gostaria que eles aparecessem na tela. Note que essa parte é apenas um rascunho, a versão real você só verá quando testar seu aplicativo num smartphone ou tablet (veremos mais adiante como fazer isso). Todos os componentes, sejam eles visíveis (botões, imagens, textos, etc) ou não (sons, sensores de movimento, acelerômetro, etc).
4. **Componentes:** lista os componentes em seu projeto. Qualquer componente que você arrastar para o Visualizador aparecerá nesta lista. Note que todo projeto começa com um componente chamado *Screen1*, que representa a primeira tela do projeto. Lembre-se que seu aplicativo sempre irá iniciar abrindo essa tela.
5. **Mídia:** Nessa área estarão listados os arquivos de mídia (imagens e sons) disponíveis para uso no seu projeto. Todo projeto começa com essa área vazia e você pode adicionar mídias a seu projeto tanto nesse ambiente quanto no ambiente Blocos.
6. **Propriedades:** Essa área lista todas as propriedades referentes ao componente selecionado na área 4. Note que as propriedades são diferentes para cada tipo de componente.

Ambiente *Blocos*

Enquanto na tela de *Designer* temos as opções para organizar a aparência de nosso projeto, no ambiente *Blocos* você irá programar o comportamento do seu aplicativo. Essa programação se dá através da concatenação de blocos de comando, por isso esse nome foi dado ao ambiente.

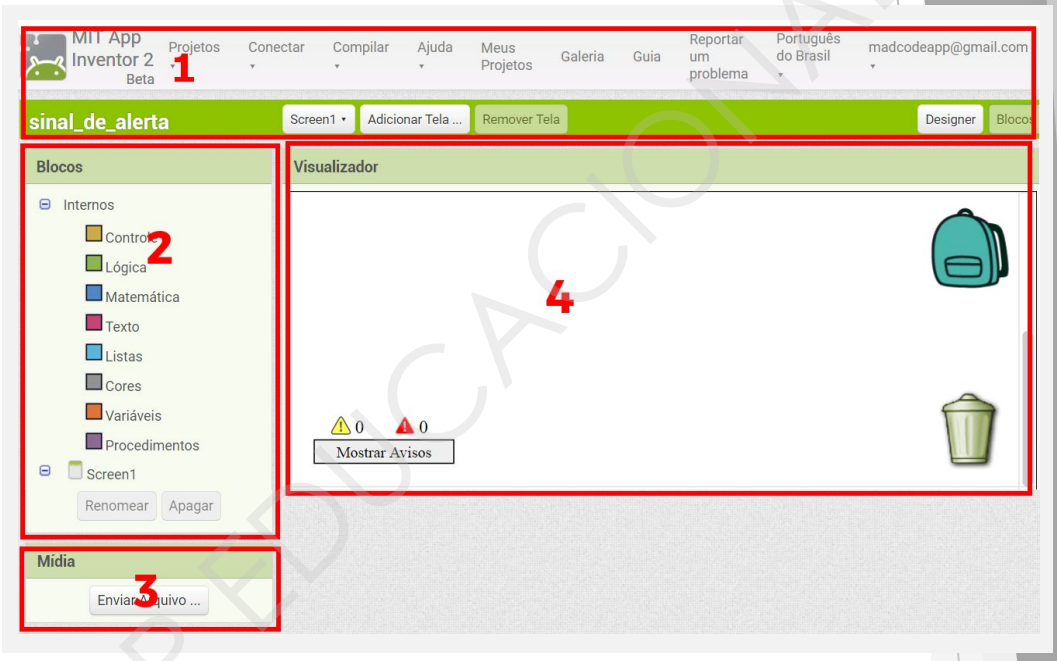


Figura 8: ambiente Blocos

O ambiente *Blocos* é dividido como você pode ver na imagem.

1. **Menu:** Contém comandos para manipular os arquivos dos aplicativos feitos, para compila-los, para construir os arquivos *.apk* (necessários para instalação nos aparelhos *Android*). Esse ambiente se mantém o mesmo tanto no ambiente *Designer* quanto no ambiente *Blocos*.

2. **Blocos:** Essa região contém os comandos que devem ser usados na programação dos aplicativos. Ela está dividida em duas seções: *Internos*, que contém os comandos que são comuns aos ambientes de programação (como ferramentas de controle de fluxo, manipulação numérica, manipulação de textos, variáveis, etc) e as de *Componentes*, que contém os comandos específicos para cada componente.
3. **Mídia:** Nessa área estarão listados os arquivos de mídia (imagens e sons) disponíveis para uso no seu projeto. Todo projeto começa com essa área vazia e você pode adicionar mídias a seu projeto tanto nesse ambiente quanto no ambiente Designer.
4. **Visualizador:** Nessa área deverão estar os blocos de comandos de programação. Todos os blocos que estiverem nessa área serão executados na ordem especificada pelo desenvolvedor. Essa área contém ainda uma *Mochila*, para guardar pedaços de código que não serão utilizados no momento atual mas que estejam nos planos para o futuro, uma *Lixeira*, para eliminar os blocos de comandos que não serão mais utilizados, e uma área para notificações sobre Avisos e Erros na Programação.

Teste de Aplicativo

Com App Inventor, você pode visualizar e testar seu aplicativo em um dispositivo Android enquanto cria. Fazer testes incrementais é uma prática utilizada por diversos desenvolvedores de software que pode economizar horas de trabalho.

Se você tiver um dispositivo Android e uma conexão à internet com Wi-Fi, você pode configurar testes ao vivo em minutos, baixando apenas um aplicativo para seu dispositivo. Se você não tem um dispositivo Android, você poderá usar um emulador para seus testes, cujas instruções para instalação podem ser encontradas (para Windows, Mac e Linux) em http://appinventor.mit.edu/explore/ai2/setup.html

Construir aplicativos com um dispositivo Android e com conexão WiFi

Esta é a opção que os desenvolvedores do App Inventor recomendam. Os passos são os seguintes:

1. Programe e compile o aplicativo no computador;
2. Envie-o a um dispositivo Android via QR Code ou por um arquivo do tipo .apk;
3. Faça o download e instalação do aplicativo em seu dispositivo, como se fosse um aplicativo comercial;
4. Execute-o no dispositivo Android.

O mais interessante desta opção é que você terá a grande motivação de ver a sua produção funcionando em seu celular, como se fosse um aplicativo profissional. É também a opção que menos depende de elementos externos ao programa, como Firewalls e programas antivírus.

AI companion

Existe uma opção mais rápida para você testar o aplicativo que fez em seu dispositivo, sem necessitar instalá-lo. Esta opção também dispensa a necessidade de instalação do emulador em seu computador. O processo é bem simples:

1. Baixe e instale em seu dispositivo Android o aplicativo MIT AI2, que você encontrará na PlayStore da Google;
2. Programe o seu aplicativo no computador com o App Inventor;
3. Clique na opção Connect / AI Companion na barra de menu do App Inventor;
4. Abra o aplicativo MIT AI2 em seu dispositivo Android e opte por scan QR code;
5. Aproxime seu dispositivo do QR Code gerado pelo App Inventor.

Criar aplicativos com um *Chromebook*

Muitos Chromebooks são capazes de executar aplicativos Android. Isso permite que você crie e execute o aplicativo no mesmo dispositivo.

Emulador

Esta opção necessita de instalação do software emulador em seu computador e, consequentemente, de direitos de instalação. Caso você opte por trabalhar com emuladores, faça, portanto, a instalação, e os devidos testes, seguindo a orientação do link enviado, antes do início da sua primeira aula.

Cabo USB

Sem WiFi? Crie aplicativos com um dispositivo Android e cabo USB. Alguns firewalls não permitem o tipo de conexão WiFi necessário para a transmissão dos arquivos. Se o WiFi não funcionar para você, tente o USB.

UNIDADE 1

AMBIENTE DE PROGRAMAÇÃO APP INVENTOR

PROJETO 'S.O.S'

PROJETO 'S.O.S'

Contexto

A proposta básica do *App Inventor* é permitir que programadores iniciantes sejam capazes de criar, com certa facilidade, aplicativos úteis. Neste sentido, ele é mais prático e menos lúdico do que o Scratch, por exemplo. Assim, este curso procurará motivar os aprendizes a programar e a pensar em aplicativos que ainda não existem, mas que podem ser uteis para algum grupo de pessoas.

O primeiro projeto, portanto, será um aplicativo pensando em pessoas que vivem em locais onde haja muitos deslizamentos de terra e, consequentemente, riscos de soterramento. A ideia básica é que o aplicativo, quando executado, emita um som de alerta pedindo ajuda.

Entre, portanto, com sua conta no App Inventor, clique em "Start New Project" e escolha o nome "SOS". Em seguida, certifique-se que o botão "Designer" esteja acionado.

Ao criar aplicativos usando o App Inventor, precisamos criar seu layout básico antes de programar. Isso acontece porque a programação nessa ferramenta é baseada em eventos que são disparados através de iterações entre o usuário e os elementos do aplicativo. Assim, não faz sentido programar o comportamento de um Button, por exemplo, sem saber se ele de fato existe e onde estará posicionado na tela. Por isso, ao longo de nosso curso, iremos falar primeiro sobre alguns componentes para, em seguida tratar de sua programação.

Neste capítulo, mostraremos 3 componentes básicos: Label, Button e Sound.

Componentes

1. Componente LABEL (menu 'interface do usuário')

Um componente Label no App Inventor é usado para exibir textos escritos na tela. Cada Legenda contém um texto pré-determinado (que pode ser vazio) inicial, que pode ser alterado caso necessário. Para adicionar uma Legenda ao seu projeto, clique em Legenda na seção Interface de Usuário e arraste para a área de Visualização. Se estiver correto, aparecerá um novo item em sua Lista de Componentes, como na imagem abaixo:

Figura 9: componente LABEL

Note que a nova *label* surgirá com o nome padrão (no nosso exemplo, *Label1*). É importante renomear os componentes de seu projeto para que você saiba exatamente qual a função de cada um deles. Por exemplo, se sua legenda for uma mensagem de boas vindas, você poderia chama-la de *MensagemBoasVindas*. Para renomear um componente, basta selecioná-lo e em seguida clicar no botão *Renomear* (veja figura a seguir):

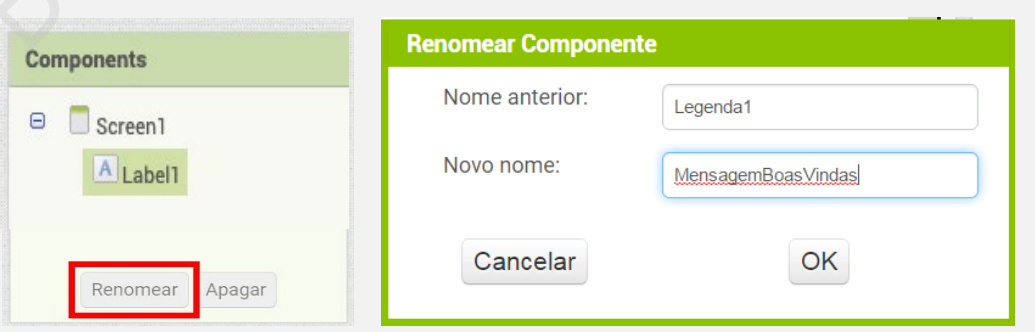


Figura 10: renomear componentes

Note que, quando se clica em um dos componentes do seu projeto, suas propriedades são apresentadas no campo "Properties".

As propriedades do componente Legenda são:

- **BackgroundColor:** Define a cor de fundo do componente;
- **FontBold:** Define se a fonte do componente estará escrita com Negrito;
- **FontItalic:** Define se a fonte do componente estará escrita com Itálico;
- **FontSize:** Define o tamanho da fonte do componente;
- **FontTypeface:** Define a família da fonte do componente (o tipo de fonte);
- **HTMLFormat:** Define se o texto permite formatação HTML;
- **HasMargins:** Decide se o componente aparece com margens. Todas as quatro margens (esquerda, direita, superior e inferior) são as mesmas. Esta propriedade não tem efeito no Designer, onde as legendas são sempre apresentadas com margens;
- **Height:** Define a altura do espaço disponível para o componente;
- **Width:** Define a largura do espaço disponível para o componente;
- **Text:** Define o texto que aparecerá escrito no componente; Note que essa é a única maneira de editar esse texto no seu projeto. Não é possível alterá-lo no *Visualizador*;
- **TextAlignment:** Indica o alinhamento (à esquerda, à direita ou centralizado) do texto do componente na direção horizontal;
- **TextColor:** Define a cor do texto do componente;
- **Visible:** Define se o componente está visível ou não.

Todas essas propriedades podem ser alteradas livremente para se adequarem a seu projeto. Os valores indicados nessa área são os valores iniciais das propriedades, ou seja, os valores com os quais cada uma das propriedades começará quando inicializarmos o nosso aplicativo. Esses valores podem ser alterados através de comandos na tela de Blocos, que veremos mais a frente.

2. Componente BUTTON (menu 'interface do usuário')

Um Button (botão) é um componente que tem a capacidade de identificar toques feitos em sua superfície. Apesar de instintivamente pensarmos nos botões físicos (por exemplo os botões de um elevador ou as teclas de um teclado), um botão para o App Inventor pode ser qualquer elemento de quem desejamos obter alguma reação quando encostado. Por exemplo, uma miniatura de uma imagem num aplicativo de fotografias que ao ser tocada aumente de tamanho poderia ser desenhada com um botão para o App Inventor.

Um button pode ser adicionado a seu projeto da mesma maneira que o componente *Legenda*, ou seja, clicando nele na Seção correta da Paleta (nesse caso, *Interface de Usuário*), e arrastando-o para o *Visualizador*.

As propriedades do componente *Botão* são:

- **BackgroundColor:** Define a cor de fundo do componente;
- **Enabled :** Define se o componente está ativado, ou seja, se ele deve ou não processar as interações dos usuários;
- **FontBold:** Define se a fonte do componente estará escrita com Negrito;
- **FontItalic:** Define se a fonte do componente estará escrita com Itálico;
- **FontSize:** Define o tamanho da fonte do componente
- **FamiliaDaFonte:** Define a família da fonte do componente (o tipo de fonte);
- **Height:** Define a altura do espaço disponível para o componente;

- **Width:** Define a largura do espaço disponível para o componente;
- **Image:** Define que imagem deverá ser usada como pano de fundo para o componente;
- **Shape**: Define o formato do Button. Existem as seguintes possibilidades:
  - Default: Padrão;
  - Rounded: Cantos arredondados;
  - Rectangular: Retangular;
  - Oval: Oval;
- ***ShowFeedback***: Especifica se um *feedback* visual deve ser mostrado quando um *Button* com uma *image* atribuida é pressionado.
- **Text:** Define o texto que aparecerá escrito no componente. Note que essa é a única maneira de editar esse texto no seu projeto. Não é possível alterá-lo no Visualizador;
- **TextAlingment:** Indica o alinhamento (à esquerda, à direita ou centralizado) do texto do componente na direção horizontal;
- **TextColor:** Define a cor do texto do componente;
- **Visible:** Define se o componente está visível ao usuário ou não.

3. Componente SOUND (menu 'mídia')

Diferente dos anteriores, esse componente não está na área User Interface, mas em *Media*. Ele é um componente invisível, isso é, não aparecerá na tela do dispositivo.

Trata-se de um componente de multimídia que reproduz arquivos de som e, opcionalmente, vibra pelo número de milissegundos (milésimos de segundo) especificado no Editor de blocos. O nome do arquivo de som a ser reproduzido pode ser especificado no *Designer* ou no Editor de blocos.

Este componente é melhor para arquivos de som curtos, como efeitos sonoros, enquanto o componente *Player* é mais eficiente para sons mais longos, como músicas.

As propriedades do componente SOUND são:

- **MinimumInterval:** O menor intervalo (em milissegundos) entre duas execuções desse som. Quando o componente tocar um som, o aplicativo bloqueia novas execuções pelo tempo destacado nessa propriedade;
- **Source:** O nome do arquivo sonoro correspondente a esse componente. Nem todos os formatos de áudio são suportados pelo App Inventor. Entre os mais conhecidos que são suportados estão o MP3, FLAC e MIDI.

Programação - Modelagem IPO

Todos os programas de computador têm uma estrutura semelhante, e os aplicativos não são diferentes. Eles são compostos de 3 partes principais: Entrada, Processamento e Saída, no que chamamos de modelo IPO (*Input, Processing & Output*).

- Um ***Input (entrada)*** para um programa são as informações que ele recebe para trabalhar. Por exemplo, a entrada pode ser uma informação digitada pelo usuário, ou um arquivo que o aplicativo obtém pela internet. Um programa também pode receber como entrada informações advindas de outros programas.
- O ***Processamento*** é a etapa em que o aplicativo utiliza as informações obtidas na entrada e as modifica de acordo com as instruções que ele recebe. O processamento pode ser simples, com apenas uma instrução, ou extremamente complexo.
- Já o ***Output (saída)*** é a etapa em que o aplicativo devolve a informação processada, seja para o usuário ou para outro aplicativo.

Essas 3 etapas normalmente acontecem em sequência, por isso uma outra maneira de enxergar esse modelo é como uma receita de bolo. As Entradas são os ingredientes, o Processamento é toda a fase de preparação e a Saída é o bolo pronto. Isso ficará mais claro conforme formos avançando nos nossos projetos.

Tomando nosso aplicativo da situação problema como base, como seria sua divisão? A tabela abaixo mostra uma divisão possível das etapas do programa :

ENTRADAS	PROCESSAMENTO	SAÍDAS
1. Toque do usuário no Botão 2. Texto inicial da Legenda 3. Arquivo do Som	1. Ao receber o toque do usuário, o botão envia uma mensagem aos componentes Legenda e Som 2. Ao receber a mensagem do Botão, o componente Legenda altera sua propriedade "Texto" 3. Ao receber a mensagem do botão, o componente Som inicia seu procedimento "Tocar"	1. O componente Legenda exibe sua mensagem alterada 2. O componente Som toca seu arquivo

Figura 11: programação IPO

Note que o usuário é responsável por apenas uma das Entradas, a de número 1. As outras são dadas pelo próprio sistema. Apesar de ter sido o programador o responsável por colocar o arquivo de Som no sistema, o usuário (ou seja, a pessoa que está manuseando o aplicativo) não tem como alterá-lo, tornando-o um pedaço do sistema.

Tendo esse planejamento em mãos, a programação ficará mais fácil, pois podemos dividir a tarefa em pequenos pedaços independentes.

O App Inventor é uma ferramenta que se baseia na programação por eventos, que são acionados por "Gatilhos". Em geral, as Entradas dos programas acontecem através desses Gatilhos.

Gatilhos

Blocos desse tipo são da cor amarela e são usados quando queremos representar uma ação inicial, em geral causada por uma interação do usuário com um componente. Todo aplicativo criado no App Inventor deve ter pelo menos um Gatilho.

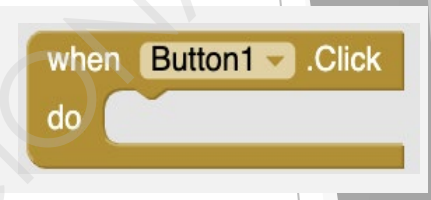


Figura 12: gatilhos

Propriedades

Blocos verdes representam as propriedades dos elementos, e são usados quando queremos alterar (verde escuro) ou obter (verde claro) alguma propriedade.

Figura 13: propriedades

Esses blocos são usados na etapa de Processamento para alterar seus valores, na etapa de Entrada para lê-los e na etapa de Saída para exibi-los.

Procedimentos

Blocos de procedimentos representam Saídas, pois tratam de ações que são feitas pelos componentes.

Figura 14: procedimentos

Programando nosso App

Os **componentes** necessários para nosso aplicativo serão um botão, uma legenda e um som, dispostos como na imagem abaixo:

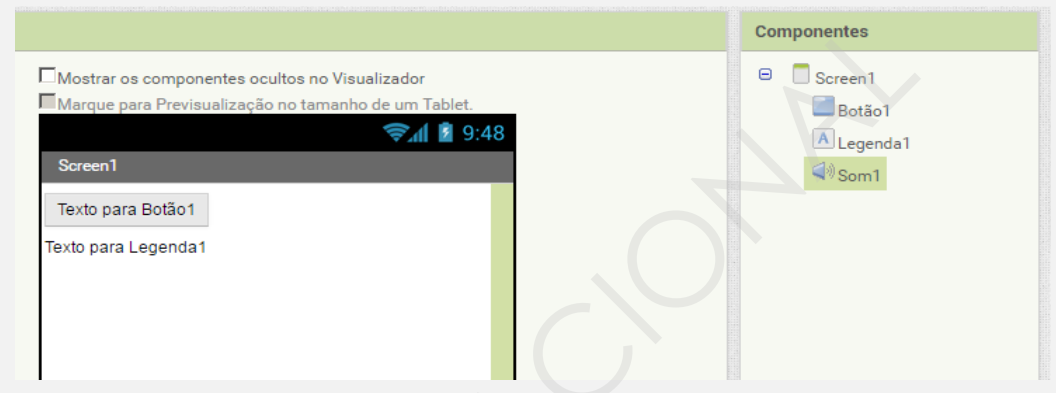


Figura 15: componentes

Você pode alterar as propriedades dos componentes como preferir. Deixamos abaixo um exemplo caso queira se inspirar, onde você pode ver a tela finalizada e as propriedades dos componentes Botão e Legenda.

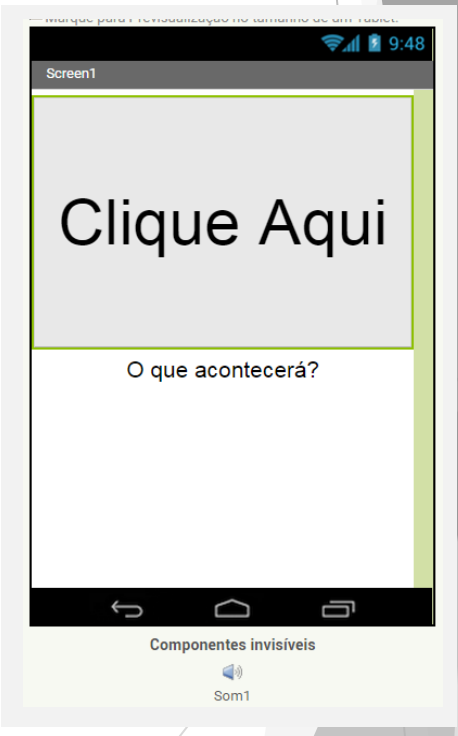


Figura 16: tela finalizada

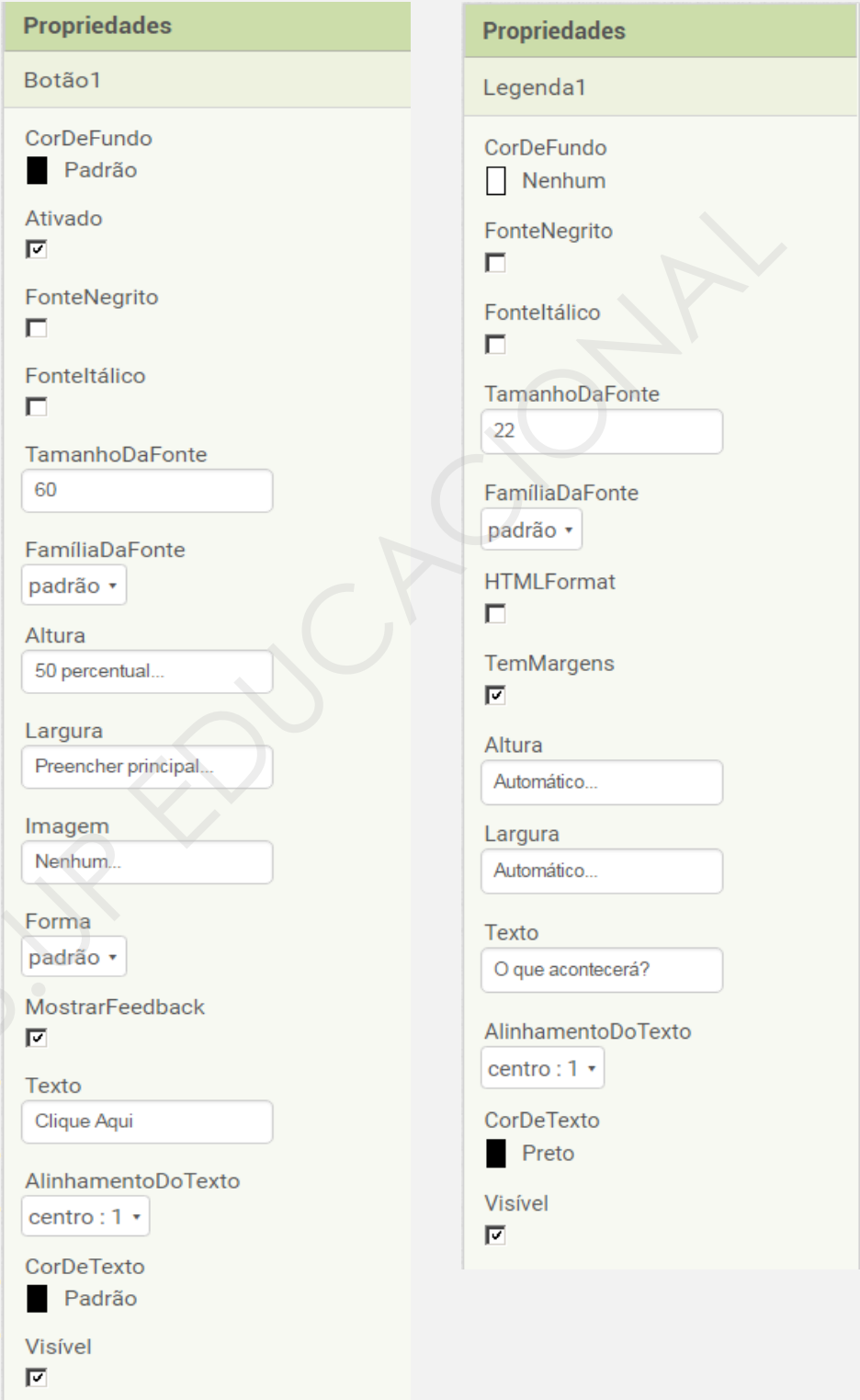


Figura 17: propriedades

Agora que você conhece os componentes necessários, podemos programar nosso App! Comece colocando o **botão**, e alterando suas dimensões para ocupar toda a tela na largura e 80% da tela na altura. Ela ficará assim:

Figura 18: botão com dimensões alteradas

Além disso, podemos adicionar uma legenda no aplicativo, para que o usuário saiba que seu clique no botão funcionou. Inicialmente, essa legenda deverá conter texto vazio. Finalmente, precisamos adicionar um componente Som, responsável por guardar o arquivo sonoro. Ao final, nosso aplicativo deverá ter a uma lista de componentes como ao lado.

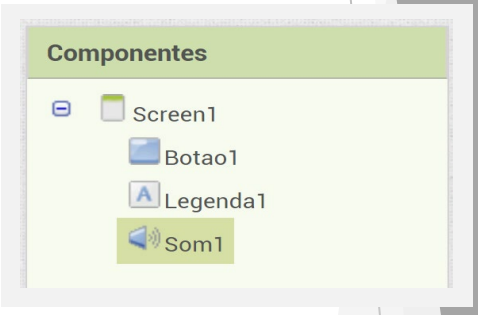


Figura 19: adicionando o componente som

Segundo nosso planejamento, ao clicar no Botão1, queremos que duas coisas aconteçam: a Legenda1 se altere (para indicar que o botão foi pressionado) e o Som seja tocado. Assim, a construção dos comandos deve englobar esses três fatores, como podemos ver abaixo:

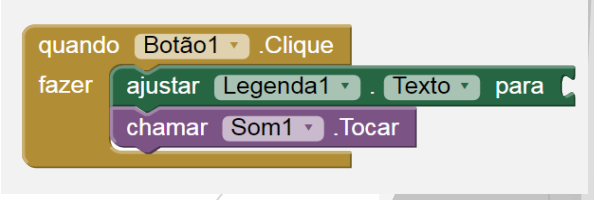


Figura 20: blocos de programação

Note que ao final do bloco verde há um encaixe. Ele serve para que completemos o comando, informando para qual valor a propriedade Texto será alterada. Para definir essa mudança, utilizaremos um bloco do menu Texto, localizado à esquerda como podemos ver na imagem abaixo. Dentro desse menu, selecionamos a opção de palavra vazia (assinalada na imagem), que iremos preencher conforme desejarmos.

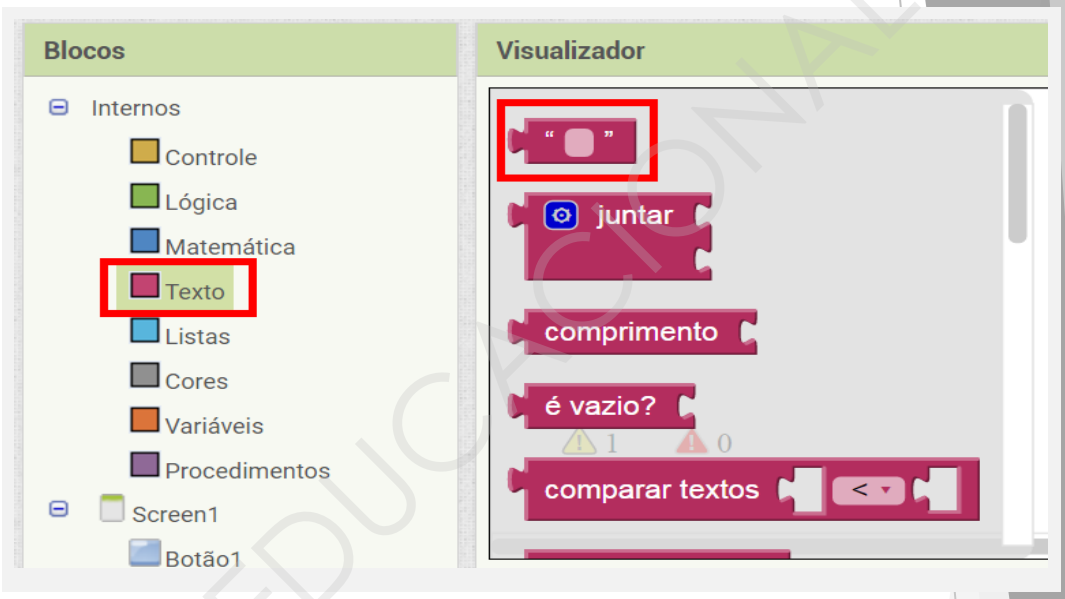


Figura 21: blocos de comando

Assim, nossa programação completa ficará como na imagem abaixo:

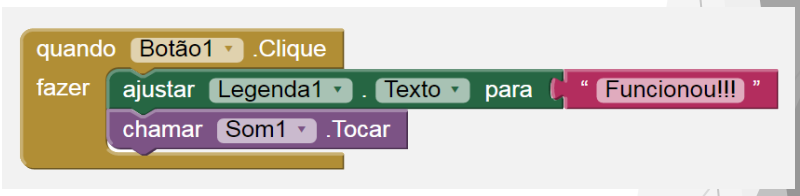


Figura 22: blocos de programação

Mas o que o Componente Som irá tocar? Para definir isso, suba um arquivo sonoro (mp3) para o App inventor através da aba de Mídia (veja a imagem ao lado):

Figura 23: upload de arquivo Mp3

Em seguida, selecione o arquivo no atributo Fonte do Componente Som:

Figura 24: seleção de arquivo

E está pronto nosso primeiro aplicativo! Agora, seu próximo passo é testá-lo! Quando fizer isso, observe bem os momentos em que acontecem a Entrada, o Processamento e a Saída.

Se você tiver um dispositivo Android e uma conexão à internet com Wi-Fi, você pode configurar testes ao vivo em minutos, baixando apenas um aplicativo para seu dispositivo. Se você não tem um dispositivo Android, você poderá usar um emulador para seus testes, cujas instruções para instalação podem ser encontradas (para Windows, Mac e Linux) em http://appinventor.mit.edu/explore/ai2/setup.html

Para testar usando o AI Companion, por exemplo, basta clicar em Assistente AI (como abaixo). Ele irá gerar um código QR, que pode ser lido pelo aplicativo AI Companion instalado no seu Android. Esse procedimento funciona se ambos os dispositivos (Computador e Smartphone) estiverem conectados na mesma rede wireless.

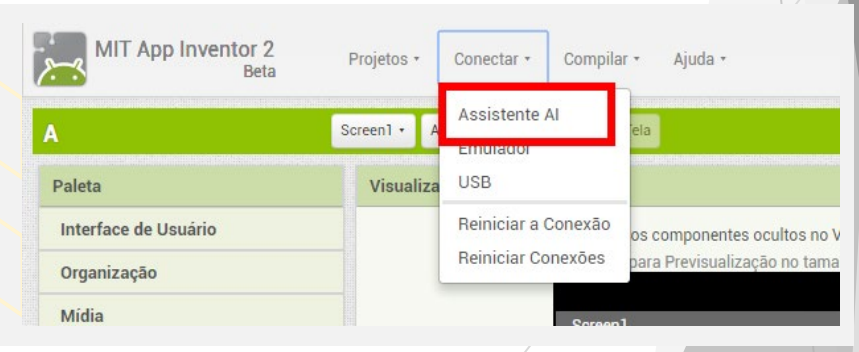


Figura 25: assistente AI Companion

Você também pode baixar direto para seu Android o seu app. Nesse caso, o aplicativo fica instalado em seu aparelho, e você pode inclusive instalar nos dispositivos de seus amigos e familiares.

Para isso, clique em Compilar, em seguida em fornecer o QR code para o .apk. Novamente, o App Inventor irá gerar um código QR que pode ser lido pelo seu celular, que automaticamente instalará o aplicativo.

Neste processo, basta seu celular estar conectado à internet, não necessariamente na mesma rede que o computador.

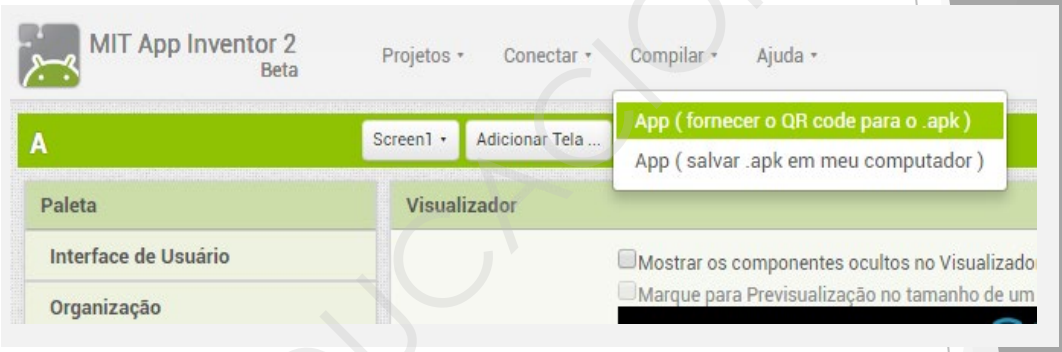


Figura 26: acesso ao app salvo

Exercícios

Chegou a hora de praticar! Siga os passos abaixo e salve seu trabalho.

1. Altere o aplicativo feito nesse capítulo para que tenha dois botões. O primeiro deve mudar o texto da Legenda e o segundo, deve tocar o Som.

2. Crie um botão que altera a cor do texto da Legenda.

3. Crie um botão que aumenta a fonte do texto da Legenda.

Desafio extra

Agora que você conseguiu criar o aplicativo pedido, vamos alterá-lo. Crie 2 botões: um que toque o alarme no volume normal e outro que toque o alarme no volume máximo.

UNIDADE 2

MODELAGEM IPO, COMPONENTES E COORDENADAS

PROJETO 'DESENHO LIVRE'

PROJETO 'DESENHO LIVRE'

Contexto

Você recebeu um pedido para criar um aplicativo que simule uma "Lousa Mágica", um brinquedo criado na década de 1950 pelo técnico em eletricidade francês Andre Cassagnes. O brinquedo, que entrou para o hall da fama dos brinquedos em 1998, é uma lousa em que você pode desenhar livremente, e pode apagar seu desenho apenas sacudindo o aparelho. Seu desafio é recriar a lousa mágica, numa versão mais moderna e com 4 cores à disposição de quem a estiver usando!

Nesta aula vamos nos aprofundar na área da programação. Nossa modelagem IPO será mais complexa, com mais Entradas e Procedimentos. Além disso, seremos apresentados a componentes de Layout e Sensores, que ampliarão nossa gama de possibilidades dentro da ferramenta. Finalmente, nesse aplicativo, o conceito de Coordenadas Cartesianas será utilizado, e daremos uma breve explicação sobre sua aplicação no App Inventor.

Componentes

1. Componente CANVAS (menu 'desenho e animação')

Uma das características mais presentes em qualquer aplicativo é a possibilidade de tocarmos em qualquer lugar da tela para obter algum resultado. Você deve ter notado em nosso aplicativo anterior que isso não necessariamente acontece sempre. Se você tocar em qualquer área que não a do componente Botão, nada acontecerá naquele aplicativo. Isso porque não utilizamos o componente Canvas, presente no menu Desenho e Animação.

A Canvas é uma grade retangular bidimensional sensível ao toque. É como um pedaço de papel no aplicativo em que você pode desenhar. Você também pode colocar imagens sobre ele e animá-las.

Podemos desenhar e colocar objetos na Canvas usando um plano de coordenadas X-Y que é semelhante ao utilizado em Matemática. A tela é composta de pixels - os menores pontos que podem aparecer em uma tela. A localização de cada pixel é definida por um par X-Y de coordenadas. X define um local no plano horizontal (a partir de 0 na extrema esquerda e aumentando à medida que você se move para a direita através da tela), e Y define uma localização no plano vertical (começando em 0 no topo e aumentando à medida que você se move a tela para baixo).

O canto superior esquerdo da Pintura começa com 0 para ambas as coordenadas (X = 0, Y = 0). Conforme você se move para a direita, a coordenada X aumenta; conforme você move para baixo, a coordenada Y aumenta.

As propriedades do componente Canvas são:

- **BackgrundColor:** define a cor de fundo do componente.
- **BackgroundImage:** define a imagem a ser usada como pano de fundo para o componente
- **FontSize:** define o tamanho da fonte do componente
- **Height:** define a altura do espaço disponível para o componente.
- **Width:** define a largura do espaço disponível para o componente.
- **LineWidth:** define a largura padrão (em pixels) das linhas desenhadas na Pintura.
- **PaintColor:** define a cor padrão das linhas desenhadas no componente.
- **TextAlingment:** indica o alinhamento (à esquerda, à direita ou centralizado) do texto do componente na direção horizontal.
- **Visible:** define se o componente está visível ao usuário.
- **ExtendMovesOutsideCanvas**: Determina se os movimentos podem se estender além das bordas da tela. O padrão é falso.

2. Componente ACCELEROMETER SENSOR (menu 'sensores')

Cada vez mais empregado nos aparelhos eletrônicos portáteis, o acelerômetro possibilita avaliar a posição relativa do aparelho e ajustar o visor do celular. Sempre que alteramos nossa posição relativa do aparelho eletrônico de retrato para paisagem, o componente acelerômetro recebe essa informação (ou seja, essa é sua Entrada) e a repassa para o aplicativo que estamos trabalhando no momento (seu Processamento). Para usar essa capacidade em nosso aplicativo, devemos usar o componente Accelerometer Sensor, localizado no menu Sensores.

- **xAccel:** 0 quando o telefone está em repouso em uma superfície plana, positivo quando o telefone está inclinado para a direita (ou seja, seu lado esquerdo está levantado) e negativo quando o telefone está inclinado para a esquerda (ou seja, seu tamanho certo é levantado).
- **yAccel:** 0 quando o telefone está parado sobre uma superfície plana, positivo quando a parte inferior está levantada e negativo quando a parte superior está levantada.
- **zAccel:** igual a -9,8 (gravidade da terra em metros por segundo) quando o dispositivo está em repouso paralelo ao solo com a tela voltada para cima, 0 quando perpendicular ao solo e +9,8 quando voltada para baixo. O valor também pode ser afetados pela aceleração com ou contra a gravidade.

As propriedades do componente Accelerometer Sensor são:

- **Enabled:** indica se o sensor inicia o aplicativo já ativo ou não
- **MinimumInterval:** indica o menor intervalo (em milissegundos) entre duas leituras do componente
- **LegacyMode**: quando habilitada, esta propriedade deixa de detectar automaticamente tablets no modo Paisagem e realizar a compensação em relação a dispositivos que normalmente estão em Retrato, como celulares. Sugere-se, portanto, que ela esteja sempre desabilitada
- **Sensitivity:** indica o quão sensível é o sensor. As escolhas possíveis são: Fraco, Moderado e Forte.

3. Componente HORIZONTALARRANGEMENT e VERTICAL ARRANGEMENT

Alguns componentes do App Inventor não são para interação com o usuário nem tem a ver com a programação dele, mas apenas servem para melhorar o design do aplicativo. É o caso dos componentes HorizontalArrangement e VerticalArrangement, que estão no menu Organização. Trata-se de um espaço na tela em que os componentes podem ser colocados em seu interior. Ao fazê-lo, os componentes todos ficam lado a lado (no caso horizontal) ou um abaixo do outro (no caso do vertical).

As propriedades de ambos os componentes são as mesmas:

- **AlignHorizontal:** define se os elementos dentro desse componente serão alinhados à esquerda, ao centro ou à direita
- **AlingVertical:** define se os elementos dentro desse componente serão alinhados no topo, ao centro ou abaixo
- **BackgroundColor:** define a cor de fundo do componente.
- **Height:** define a altura do espaço disponível para o componente.
- **Width:** define a largura do espaço disponível para o componente.
- **Image:** define a imagem a ser usada como pano de fundo para o componente
- **Visible:** define se o componente está visível ao usuário ou não.

Agora que entendemos quais serão os "ingredientes" que utilizaremos em nossa "receita", vamos ao modo de preparo, ou seja, vamos ao algoritmo.

Programação - Modelagem IPO

Diferentemente do aplicativo do projeto anterior, esse possui diversas formas de interação com o usuário. Note que um usuário poderia utilizar apenas um botão durante todo seu uso do aplicativo, ignorando os outros. E seu aplicativo deverá funcionar normalmente.

Em termos da modelagem IPO teremos diversos fluxos de programa, que podem ou não se encontrar. Todos esses encontros precisam estar planejados em sua modelagem, ou seja, se uma entrada implica na alteração de uma variável, por exemplo, quando uma nova entrada for realizada isso deve ser levado em consideração. Tome como exemplo o planejamento a seguir:

ENTRADAS	PROCESSAMENTO	SAÍDAS
1. Toque do usuário no Botão_troca_cor 2. Toque do usuário no botão_apagar 3. Toque do usuário na Pintura tela	1. Ao receber o toque do usuário no botão_troca_cor, o sistema deve mudar a cor atual para azul, se estiver com a cor vermelha, e vice-versa. 2. Ao receber o toque do usuário no botão_apagar, o sistema deve apagar todos os traços feitos na tela 3. Ao receber um toque na tela, o sistema deve pintar o local correspondente com a cor atual	1. A cada instante, o sistema deve exibir na tela o rastro dos toques realizados na tela, com a cor selecionada até o momento.

Figura 27: programação IPO

Note que o processamento 3 (pintura) referente à entrada 3 (toque na tela) pode variar sua cor, que pode ser alterada a partir da entrada 1. Ou seja, apesar dos processamentos serem independentes entre si, eles podem alterar o comportamento de outros.

Programando nosso App

Nesse projeto, iremos criar um aplicativo de desenho, em que o usuário pode trocar a cor (azul ou vermelho) do desenho e apaga-lo caso necessário. Para isso, iremos usar 4 componentes: 1 Canvas, 1 HorizontalArrangement e 2 Buttons, dispostos como você vê na imagem a seguir:

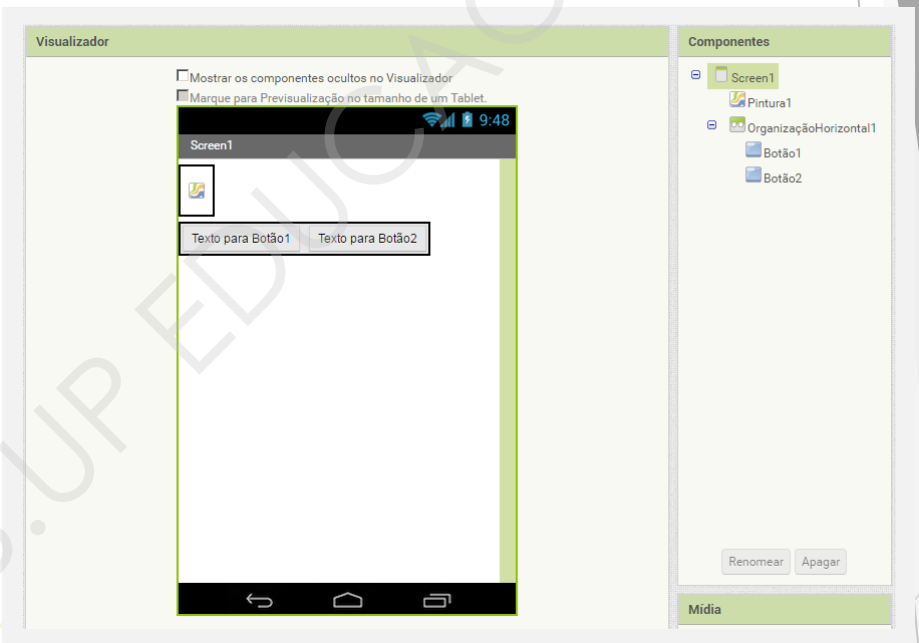


Figura 28: componentes e botões

Para que o design da tela fique mais amigável, iremos alterar as propriedades dos componentes para que fique como na imagem ao lado. Altere as propriedades em seu programa para que se assemelhe à imagem:

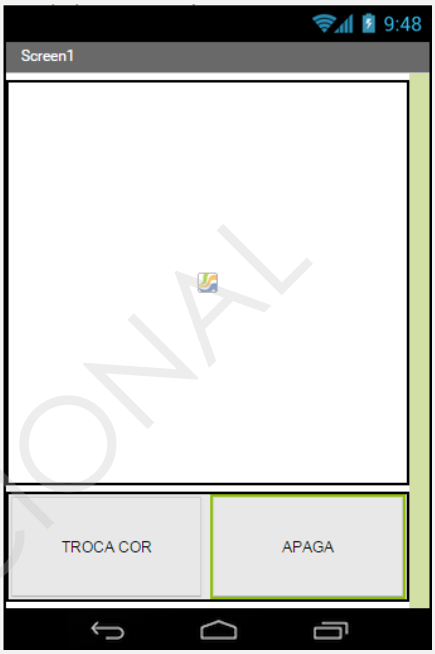


Figura 29: design da tela com componentes alterados

Finalmente, precisamos alterar o nome dos componentes em nosso aplicativo, pois da maneira como está ficaria muito confuso. Por exemplo, poderíamos nos confundir com os dois botões sem saber qual troca a cor do pincel e qual apaga o desenho. Para isso, usamos o botão de renomear, ficando como resultado final que você vê ao lado:

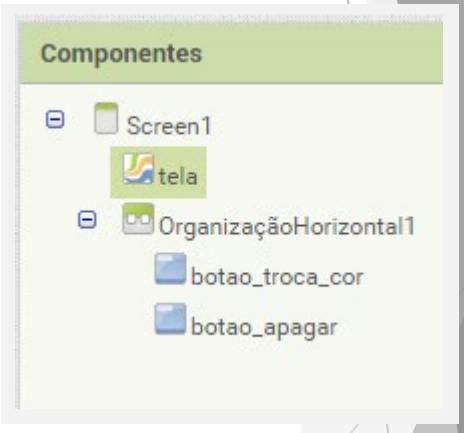


Figura 30: alterando nomes dos componentes

Vamos agora estudar as Entradas, Processamentos e Saídas desse aplicativo. Novamente, perceba como essa modelagem induz nossa programação. Note também que nosso aplicativo tem mais de um gatilho.

Vamos começar a programação pelo botão_apagar. Ao clicar nele, a tela deve eliminar todos os seus traços. Ou seja: o botão irá iniciar um processamento que interferirá nas características/propriedades do componente "tela". Assim, o comando que irá dentro do gatilho deve ser encontrado na Pintura tela, como você vê abaixo:

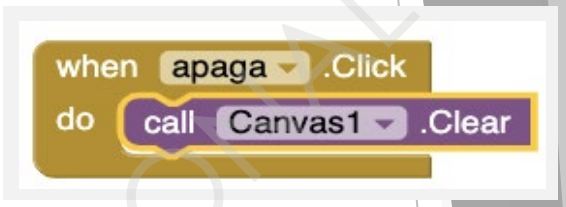


Figura 31: blocos de programação

Para programar a troca de cores, iremos utilizar um bloco de condição. Ele está no menu de Controle, localizado à esquerda como você vê a seguir:

Figura 32: blocos de programação

Nesse aplicativo, temos apenas duas opções para cores: se não for azul, deverá ser vermelho. Assim, iremos usar o bloco se_então_senão, que deve ser construído como se vê abaixo:

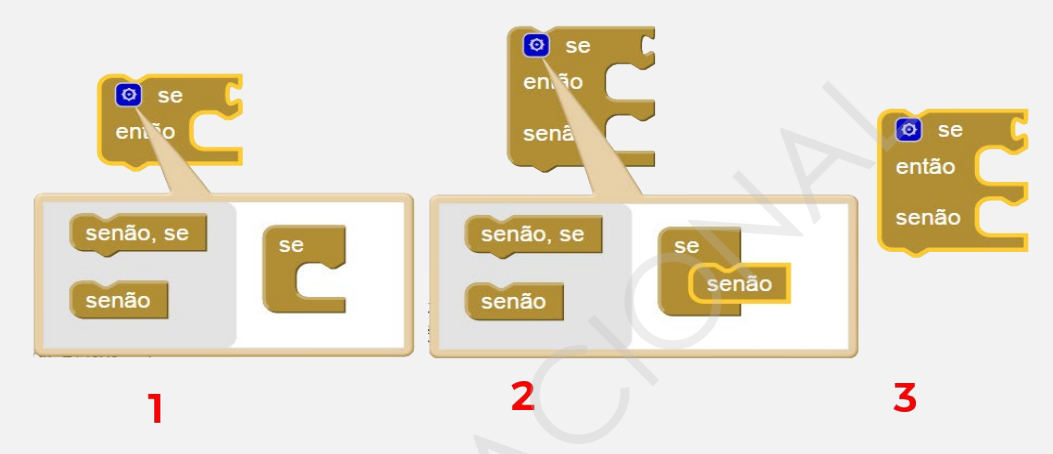


Figura 33: blocos 'se_então_senão'

Ao clicar no bloco, você verá um botão azul. Clique nele (passo 1) para abrir o menu. Clique no bloco senão e o arraste para dentro do bloco se (passo 2). Finalmente, você verá o novo bloco se então senão (passo 3).

Agora precisamos de uma maneira de marcar qual a cor atual de nosso pincel, para que ele saiba para que cor trocar ao receber o comando vindo da entrada. Para isso, iremos criar uma **variável global** que marcará a cor atual. Para criar uma variável, basta clicar no menu à esquerda, e nomeá-la como preferir.

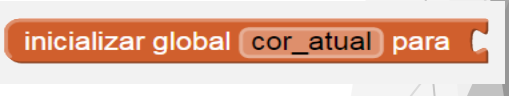


Figura 34: variável global

Essa variável será um número que representará as cores Azul e Vermelho. Para a cor Azul, escolhemos o número 0 e para a cor Vermelha, o número 1. Assim, podemos escolher inicializar qualquer uma das duas, e escolheremos a cor Azul (e o número 0), resultando no bloco abaixo:

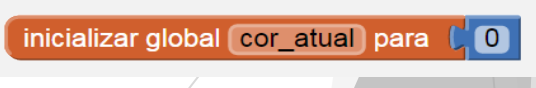


Figura 35: variável global

Finalmente, construiremos a estrutura condicional que estará dentro do Gatilho do botão_troca_cor, que você vê abaixo.

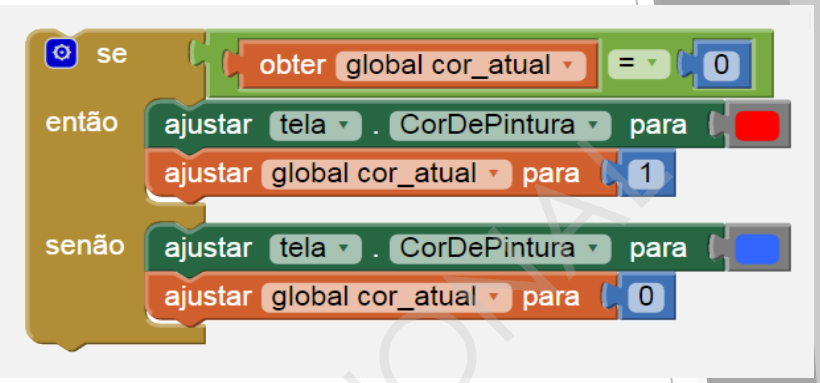


Figura 36: estrutura condicional

Vamos entender o que está acontecendo. Quando a variável cor_atual é 0, significa que a cor do pincel agora é azul. Assim, para trocá-la, precisamos selecionar a cor vermelha e atualizar a variável cor_atual para 1, pois agora nossa cor é vermelha. Finalmente, iremos colocar esse bloco dentro do Gatilho botão_troca_cor, finalizando essa segunda etapa.

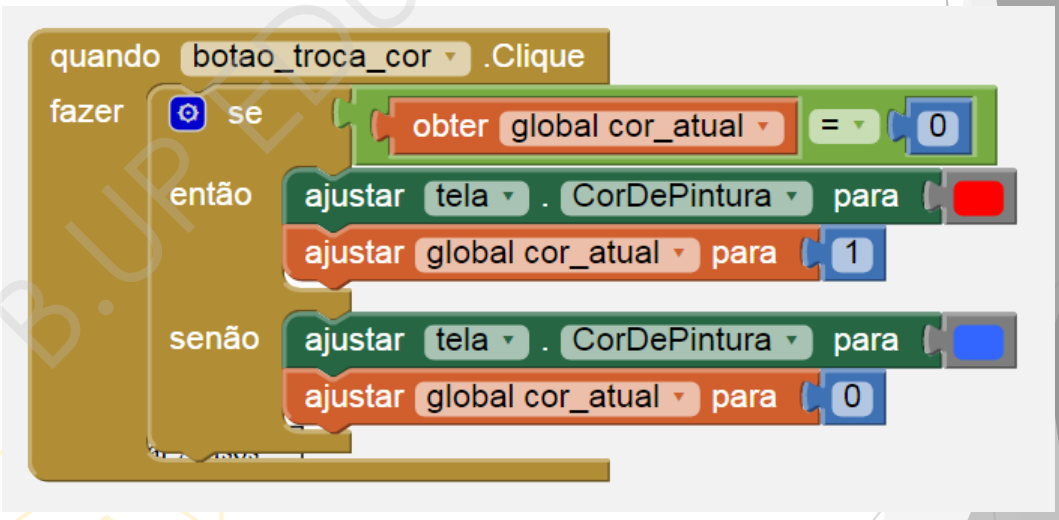


Figura 37: segunda etapa da estrutura condicional

Finalmente, vamos criar o Gatilho que desenhará os efeitos na tela. Para isso, utilizaremos um procedimento do componente Pintura chamado DesenharLinha, que cria uma linha entre 2 pontos dados. Além disso, o Gatilho que usaremos da Pintura é o Arrastado.

A estrutura básica que utilizaremos é a que podemos ver na imagem abaixo. Note que colocamos 4 variáveis preenchendo os campos do procedimento DesenharLinha.

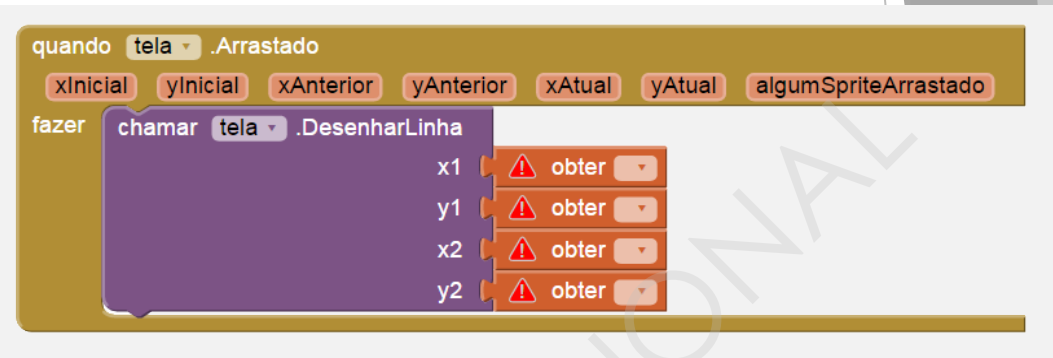


Figura 38: variáveis do gatilho 'arrastado'

Agora precisamos definir quem serão as variáveis a serem preenchidas. Clicando na lista, podemos ver as seguintes opções:

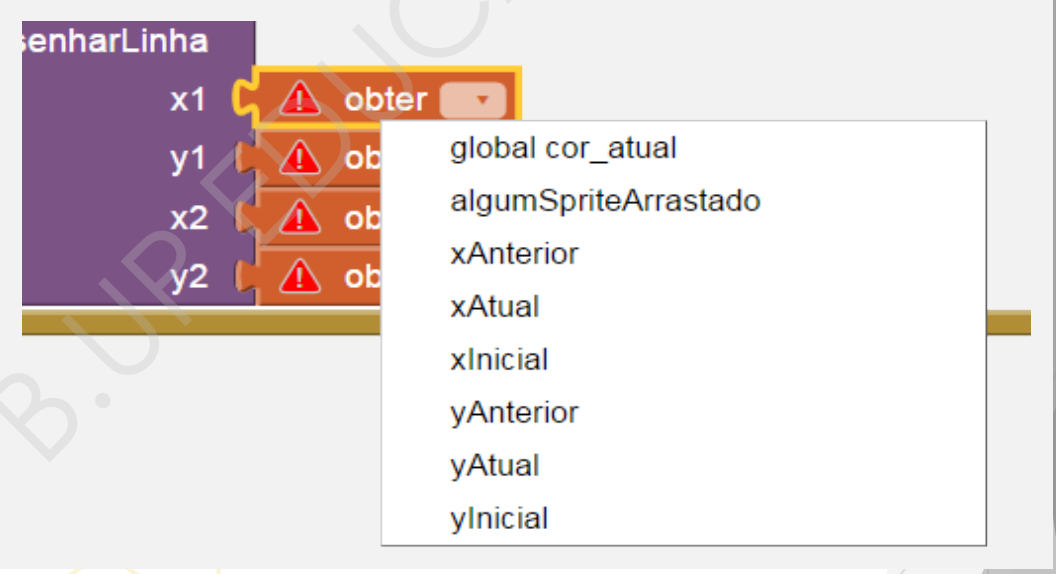


Figura 39: definição de variáveis

Ao arrastar o dedo sobre a tela, a cada instante temos uma mudança de coordenadas na tela. Assim, ao longo da execução serão feitas diversas pequenas linhas, que juntas formarão o desenho completo. Assim, a cada instante, queremos juntar o ponto anterior com o ponto atual. Assim, o resultado final da programação desse bloco será como se vê abaixo.

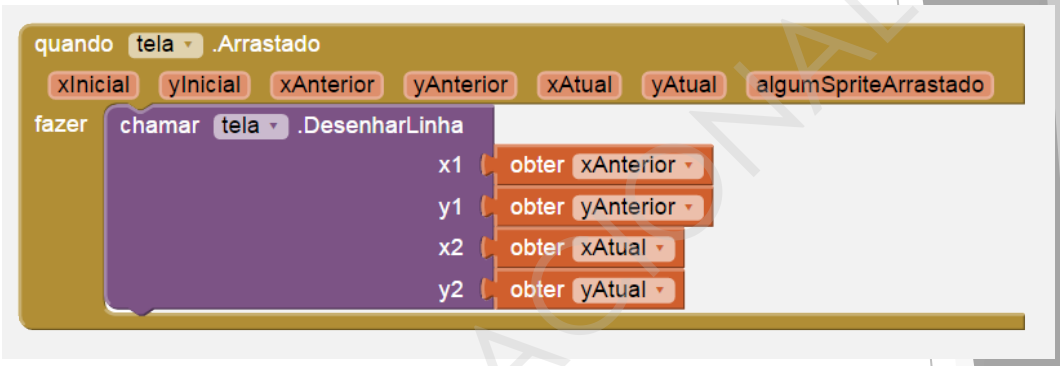


Figura 40: resultado da programação do bloco

Exercícios

Chegou a hora de praticar! Siga os passos abaixo e salve seu trabalho.

1. Modifique o aplicativo que criamos para que troque entre 3 cores.

2. Como utilizar o SensorAcelerômetro? Identifique o Gatilho correto e altere o comportamento do aplicativo para que ele faça o procedimento 'Apaga' ao sacudirmos o aparelho.

Desafio extra

Faça um desenho de arco-íris, programando para que a cor do traço se altere dinamicamente ao longo da execução do programa.

UNIDADE 3

DINÂMICA, TRANSIÇÃO E PROPRIEDADES

PROJETO 'DESENHO LIVRE 2.0'

PROJETO 'DESENHO LIVRE 2.0'

Contexto

Sua Lousa Mágica foi um sucesso! Dessa vez, a empresa pediu uma nova versão do app, com mais telas, mais opções de cores e outras funcionalidades como mudar o formato do pincel, por exemplo. Retome seu antigo projeto e encare esse novo desafio!

Nesse projeto, iremos adicionar dinâmica ao nosso aplicativo. Mudanças de tela existem em quase todos os aplicativos, e iremos fazer também. Para isso, alteraremos a propriedade "Visível" dos elementos da tela, para aparecerem somente quando a "tela" correta estiver ativa. Apesar de parecer mais complicada, essa maneira de programar torna mais simples a transição de variáveis entre as telas, por exemplo. Para exemplificar, iremos criar uma tela de seleção de cores num aplicativo semelhante à Lousa Mágica, que fizemos no começo do curso.

Componentes

Neste app, utilizaremos diversos componentes que já vimos antes. Um componente novo é o de Compartilhamento, que descreveremos abaixo.

1. Componente SHARING (menu 'social')

O Sharing é um componente não visível que permite compartilhar arquivos e/ou mensagens entre seu aplicativo e outros aplicativos instalados em um dispositivo. O componente exibirá uma lista dos aplicativos instalados que podem manipular as informações fornecidas e permitirá que o usuário escolha uma para compartilhar o conteúdo com, por exemplo, um aplicativo de e-mail, um aplicativo de rede social, um aplicativo de mensagens de texto e assim por diante. Este componente não possui propriedades.

Programando nosso App

Para esse aplicativo, utilizaremos 2 componentes Canvas. Uma delas, será a tela que será pintada, enquanto a outra representará uma paleta de cores. Vamos fazer primeiro a Canvas que será desenhada. Para tanto, colocaremos uma Canvas (ocupando 80% da altura) e uma HorizontalArrangement (ocupando os 20% restantes).

Dentro da HorizontalArrangement colocaremos 4 botões: Aumenta, Diminui, Compartilhar e Paleta. A disposição dos componentes ficará como a imagem a seguir:

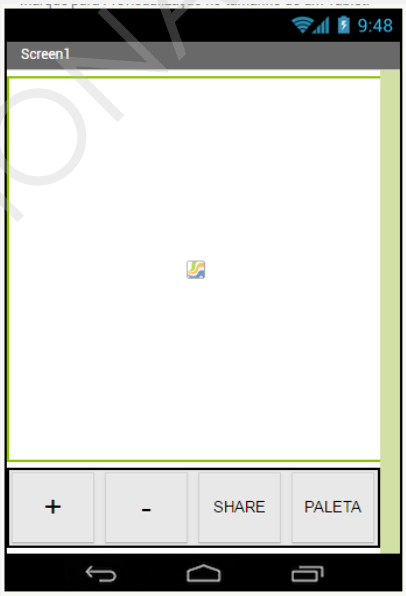


Figura 41: disposição dos componentes

Vamos programar essa parte da tela. O funcionamento básico é igual ao do app Lousa Mágica anterior. Para relembrar, colocamos a parte relevante abaixo:

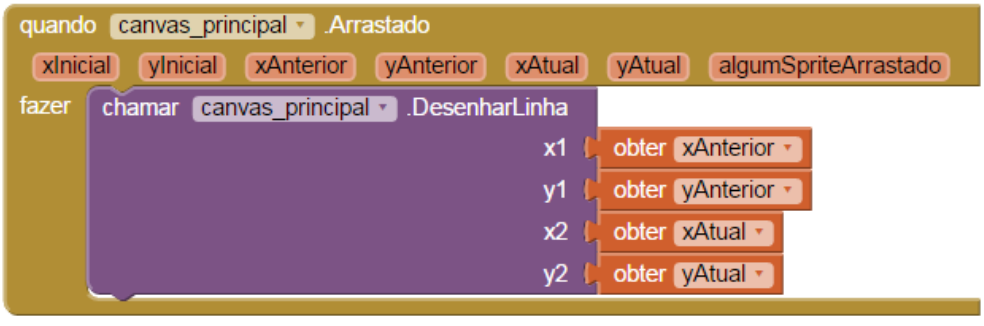


Figura 42: programação

Os botões de aumentar e diminuir (+ e -) alteram a espessura da linha desenhada. Uma maneira simples e elegante de fazer essa programação está abaixo:

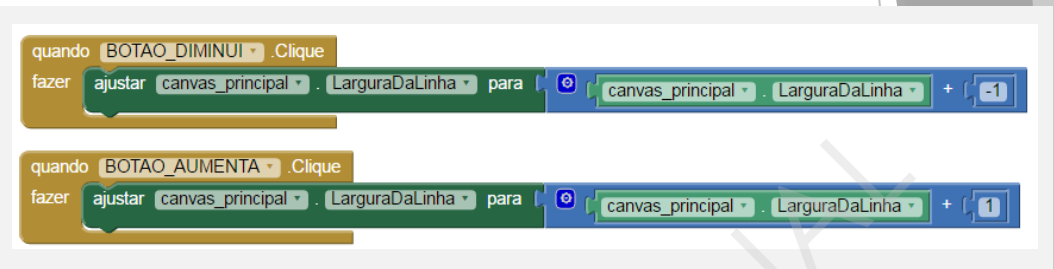


Figura 43: uso dos botões de aumentar e diminuir

Finalmente, o botão de Sharing deverá enviar uma cópia da tela para o aplicativo disponível no smartphone do usuário. Isso pode ser feito da seguinte maneira:

Figura 44: botão de compartilhamento

Esse bloco salva o canvas_principal (como se tirasse uma foto dele) como um arquivo de imagem e envia para o aplicativo de escolha do usuário, podendo ser um aplicativo de e-mail, de redes sociais, de mensagens instantâneas, etc.

Até aqui temos um aplicativo muito semelhante ao de Lousa Mágica. Agora, iremos criar a Canvas da paleta. Para isso, na Canvs atual (canvas_principal) desmarque a propriedade Visível. Faça isso também na propriedade Visível da HorizontalArrangement. Finalmente, marque a propriedade Visível da canvas_paleta, para que você possa trabalhar nela.

A paleta deverá preencher toda a tela (em ambas as dimensões). Adicione bolas de raio 40 na tela, e dê para cada uma delas cores diferentes. A seguir você vê um exemplo com 3 bolas de 3 cores.

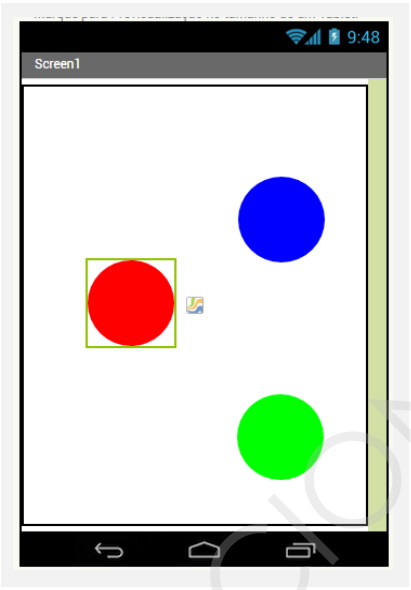


Figura 45: paleta de cores

Vamos agora programar. Ao invés de fazer um script para cada bola, faremos um código que seleciona a cor tocada pelo usuário. Assim, mesmo se adicionarmos mais bolas coloridas ao programa, não precisaremos mudar sua programação. Observe o código abaixo:

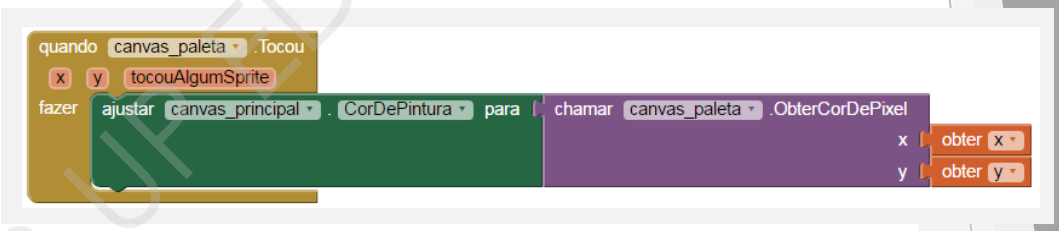


Figura 46: início da programação

Aqui, pegamos a cor do pixel tocado pelo usuário, independente de qual ele for. E para voltar para a tela anterior, podemos adicionar os códigos nesse mesmo gatilho. Assim, sempre que o usuário tocar no canvas_paleta, ele irá voltar para a ela anterior. O Gatilho completo fica assim:

Figura 47: gatilho completo

Finalmente, precisamos inicializar o aplicativo corretamente, deixando visível a tela de desenho e ajustando o tamanho inicial do pincel para 1. Além disso, precisamos programar o botão paleta para abrir a paleta. Ambos os códigos estão abaixo:

Figura 48: programação para iniciar o aplicativo

Exercícios

Chegou a hora de praticar! Siga os passos abaixo e salve seu trabalho.

1. Adicione mais botões de cores no seu programa
2. Adicione um botão de apagar, que limpa o canvas principal

Desafio extra

Ao invés de colocar bolas para as cores da paleta, experimente baixar um arquivo de imagem colorida e coloque-o como fundo de tela do canvas_paleta. O que acontece?

UNIDADE 4

PROCESSAMENTO E INSTRUÇÕES CONDICIONAIS

PROJETO 'CALCULADORA IMC'

PROJETO 'CALCULADORA IMC'

Contexto

O índice de Massa Corpórea, é uma medida internacional usada para indicar se uma pessoa está no peso ideal. Tal índice foi desenvolvido pelo matemático, sociólogo e demógrafo belga *Lambert Quételet* no fim do século XIX. Trata-se de um método fácil e rápido para a avaliação do nível de gordura de cada pessoa e é adotado pela Organização Mundial da Saúde como um preditor de obesidade. Para auxiliar a população, sua empresa pediu para que você criasse um aplicativo que calcula o IMC de uma pessoa com base em seu peso, sua altura em centímetros e seu sexo. Em seguida, deve calcular o IMC seguindo a seguinte fórmula:

$$\text{IMC} = \frac{\text{peso}}{\text{altura} \times \text{altura}}$$

Em seguida deve exibir na tela a mensagem correspondente, dependendo do resultado do cálculo, seguindo a tabela abaixo (que prevê altura em *metros* e peso em *quilogramas*):

CONDIÇÃO	IMC EM MULHERES	IMC EM HOMENS
Abaixo do peso	< 19,1	< 20,7
Dentro do peso ideal	19,1 ~ 25,8	20,7 ~ 26,4
Um pouco acima do peso ideal	25,8 ~ 27,3	26,4 ~ 27,8
Acima do peso ideal	27,2 ~ 32,3	27,8 ~ 31,1
Muito acima do peso ideal (obeso)	> 32,3	> 31,1

Tabela 1: intervalos de IMC

Até agora, você lidou com programas de várias entradas e saídas variadas. Nesse capítulo, nós focaremos nos aspectos de programação. Assim, nosso grande foco será na etapa de processamento do aplicativo, usando técnicas de programação conhecidas como "Instruções Condicionais".

Componentes

1. Componente CAIXADETEXTO (menu 'interface do usuário')

O componente *CaixaDeTexto* traz para o aplicativo um espaço onde o usuário pode digitar uma palavra, uma frase ou um número. Esse componente é usado para obter informações do usuário para a realização dos procedimentos.

As propriedades do componente *CaixaDeTexto* são:

- **BackgroundColor:** Define a cor de fundo do componente.
- **Enabled:** Define se o componente começa ativo, ou seja, se o usuário pode interagir com aquele componente no início do programa.
- **FontBold:** Define se a fonte do componente estará escrita com Negrito.
- **FontItalic:** Define se a fonte do componente estará escrita com Itálico.
- **FontSize:** Define o tamanho da fonte do componente
- **FontTypeface:** Define a família da fonte do componente, isto é, o tipo de fonte
- **Height:** Define a altura do espaço disponível para o componente.
- **Width:** Define a largura do espaço disponível para o componente.
- **Text:** Define o texto que aparecerá escrito no componente. Note que essa é a única maneira de editar esse texto no seu projeto. Não é possível alterá-lo no Visualizador.

Dica: *O texto que deve aparecer na caixa de entrada para fornecer uma dica sobre o que o usuário deve entrar. Esse texto só será visto se a caixa de texto está vazia.*

- **Multilinha:** Se for verdadeira, então esta caixa de texto aceita várias linhas de entrada, que são inseridos usando a tecla de Retorno.
- **SomenteNúmeros:** Se for verdadeira, então essa caixa de texto aceitará apenas números, ignorando a entrada de outros caracteres.
- **AlinhamentoDoTexto:** Indica o alinhamento (à esquerda, à direita ou centralizado) do texto do componente na direção horizontal.
- **CorDeTexto:** Define a cor do texto do componente.
- **Visível:** Define se o componente está visível ao usuário.

2. Componente ESCOLHELISTA (menu 'interface do usuário')

O componente EscolheLista é um botão que, quando clicado, exibe uma lista de textos para o usuário escolher. Os textos podem ser especificados através do ambiente Designer, definindo a propriedade *CadeiaDeElementos* com os itens separados por uma vírgula (por exemplo, "opção 1, a opção 2, a opção 3").

As propriedades do componente *EscolheLista* são:

- **CorDeFundo:** Define a cor de fundo do componente.
- **CadeiaDeElementos:** Define os elementos da lista que poderão ser escolhidos, separados por vírgula.
- **Ativado:** Define se o componente começa ativo, ou seja, se o usuário pode interagir com aquele componente no início do programa.
- **FonteNegrito:** Define se a fonte do componente estará escrita com Negrito.
- **FonteItalico:** Define se a fonte do componente estará escrita com Itálico.
- **TamanhoDaFonte:** Define o tamanho da fonte do componente.

- **FamiliaDaFonte:** Define a família da fonte do componente, isto é, o tipo de fonte.
- **Altura:** Define a altura do espaço disponível para o componente.
- **Largura:** Define a largura do espaço disponível para o componente.
- **Imagem:** Especifica uma imagem para o pano de fundo do componente.
- **CorDeFundoDoItem:** Especifica a cor de fundo da lista de itens.
- **CorDoTextoDoItem:** Especifica a cor da fonte da lista de itens.
- **Seleção:** Representa o item selecionado por padrão.
- **Forma:** Especifica a forma do botão (padrão, arredondado, retangular, oval).
- **Texto:** Define o texto que aparecerá escrito no componente. Note que essa é a única maneira de editar esse texto no seu projeto. Não é possível alterá-lo no Visualizador.
- **AlinhamentoDoTexto:** Indica o alinhamento (à esquerda, à direita ou centralizado) do texto do componente na direção horizontal.
- **CorDeTexto:** Define a cor do texto do componente.
- **Visível:** Define se o componente está visível ao usuário.

Programação - Modelagem IPO

A modelagem desse aplicativo é bem simples, com apenas uma entrada, um processamento e uma saída. Isso não significa que o aplicativo seja mais fácil que os anteriores, apenas que as etapas ficaram mais complexas em termos computacionais.

ENTRADAS	PROCESSAMENTO	SAÍDAS
1. Usuário insere peso e altura	1. Calcula a equação peso / altura ²	1. Imprime o resultado na tela

Figura 49: programação IPO

Programando nosso App

Vamos criar um aplicativo que calcula o IMC do usuário, lendo o peso e a altura dele. Veja abaixo a interface do aplicativo, já com os componentes listados e renomeados ao lado.

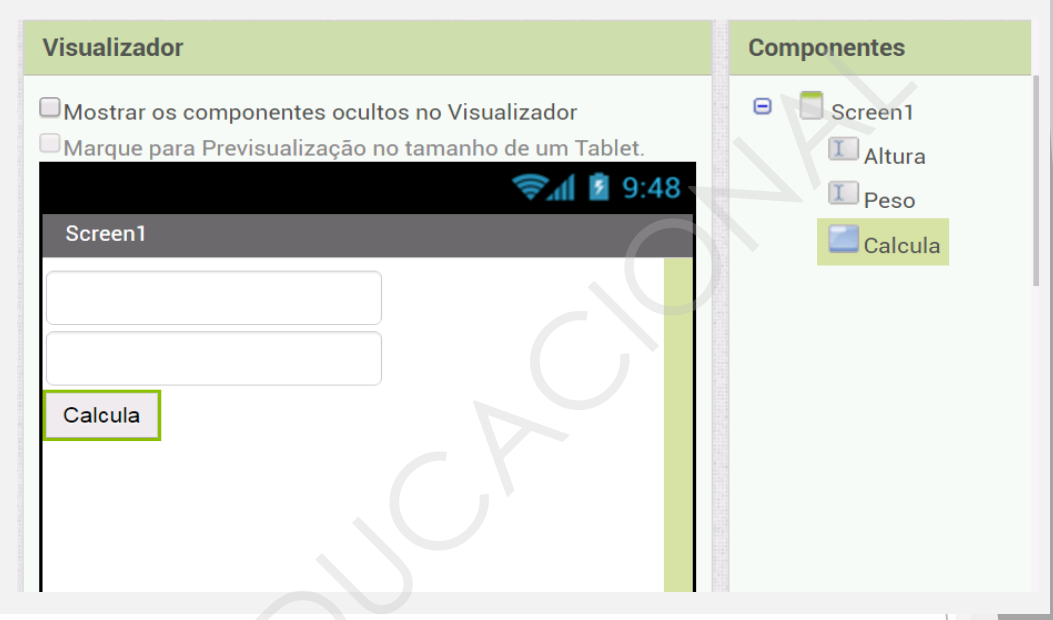


Figura 50: interface do aplicativo

Note que nosso aplicativo apenas lê a altura e o peso do usuário, sem fazer distinção do sexo do usuário. Essa distinção acontecerá mais a frente. Além disso, não fazemos uma filtragem da formatação que o usuário dará aos dados. Vamos supor, a princípio, que o usuário irá digitar o peso em quilos e a altura em centímetro.

Criaremos 2 variáveis para esse projeto, inicialmente: peso e altura. Elas serão lidas como você vê aqui:

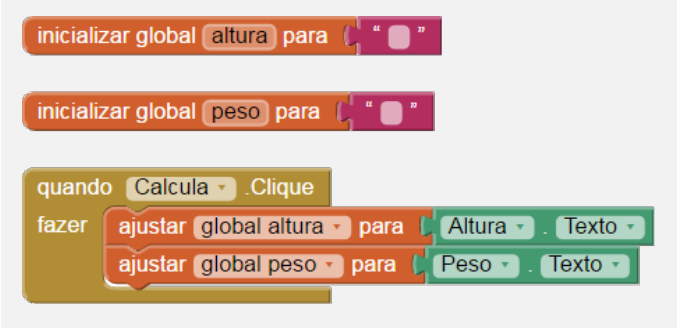


Figura 51: variáveis peso e altura

Em seguida, precisamos converter a altura que está em centímetros para metros. Para isso, usaremos os blocos do menu "Matemática". A conversão acontece como se vê abaixo (suprimimos os blocos que não são relevantes nessa operação).

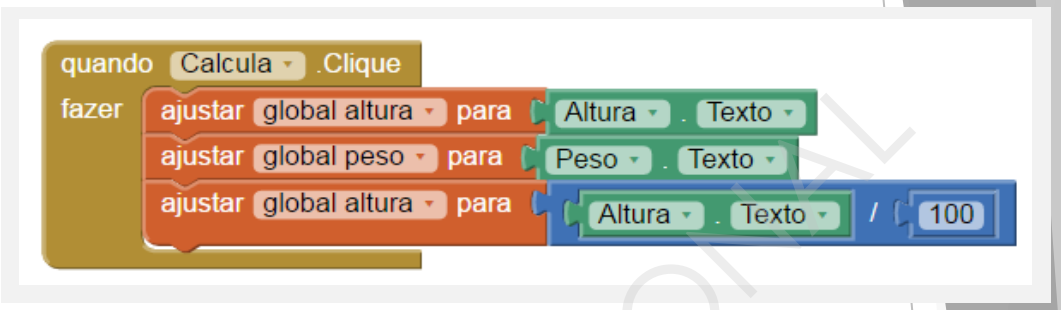


Figura 52: uso do bloco 'matemática' para conversão de medidas

Em seguida, calculamos o IMC como na fórmula apresentada no início da aula:

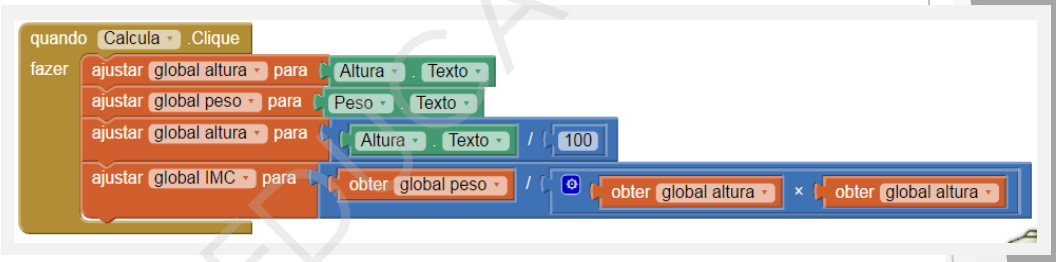


Figura 53: programando o cálculo do IMC

Dessa maneira, temos o resultado numérico esperado. Suponha agora que queiramos alertar o usuário caso seu IMC seja maior que 32.3, o que, segundo nossa tabela, seria considerado uma pessoa obesa (para homens). Para isso, precisamos buscar nos blocos de controle uma condicional, construída como abaixo:

Figura 54: uso de condicional

Esse bloco sozinho não terá efeito, pois nenhum gatilho aciona seu funcionamento. Portanto, precisamos colocá-lo dentro de um gatilho, após o cálculo do IMC ter sido realizado. Assim, ficamos com o bloco como o abaixo:

Figura 55: gatilho

Adicione agora um componente EscolheLista. Usaremos esse componente para dar opções para o usuário escolher seu gênero, uma vez que a informação sobre o IMC é diferente dependendo do sexo de nascimento da pessoa. Para adicionar as opções, faça como o bloco abaixo:

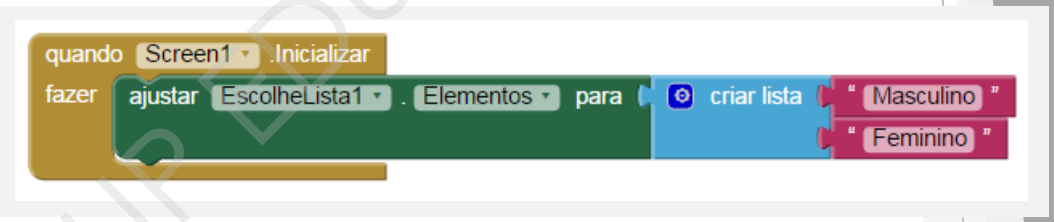


Figura 56: adição do componente EscolheLista

Em seguida, criaremos uma variável "sexo", que armazenará o sexo escolhido (homem ou mulher), conforme vemos a seguir:

Figura 57: criação da variável 'sexo'

Agora, podemos alterar o código do botão Calcula para ficar mais preciso. O software se perguntará, além do valor do IMC, o sexo definido pelo usuário, para determinar se o IMC indica obesidade ou não. O código completo (desse componente) fica como o a seguir:

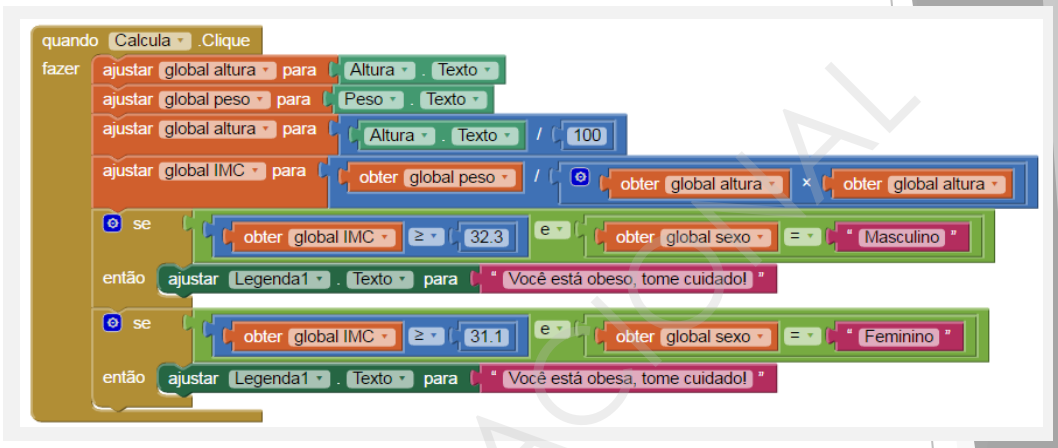


Figura 58: código completo

Note que aqui usamos um operador lógico para identificar se duas condições são verdadeiras ao mesmo tempo.

Exercícios

Chegou a hora de praticar! Siga os passos abaixo e salve seu trabalho.

1. Inclua uma legenda para informar o IMC do usuário.
2. Modifique o programa para informar que o usuário não é obeso.
3. Modifique o programa para informar quando o usuário está muito abaixo do peso segundo nossa tabela.
4. Faça o programa devolver a classificação do usuário baseando-se no sexo e na faixa de peso dele.

Desafio extra

1. Faça o Aplicativo calcular quantos quilos o usuário deve perder para que seu IMC seja considerado Ideal
2. Adicione uma terceira opção em "Gênero", por exemplo, "não quero informar". Nesse caso, o aplicativo deve considerar as estatísticas do usuário do sexo masculino como base para determinar a faixa de peso.

UNIDADE 5

APLICATIVOS DINÂMICOS E COMPONENTES

PROJETO 'TESTE DE REFLEXOS'

PROJETO 'TESTE DE REFLEXOS'

Contexto

A gamificação (ou ludificação) é um conceito que visa utilizar mecânicas de jogos em situações do cotidiano, como por exemplo em aulas regulares ou treinamento de empresas. A ideia é que a mecânica de jogo incentive as pessoas a realizar as suas tarefas da melhor maneira possível, obtendo um bom resultado de acordo com a mecânica do game.

Sua empresa foi contratada para criar um aplicativo que ajude médicos a diagnosticar problemas de atenção em crianças. Isso deverá ser feito através de um game em que diversos personagens aparecem na tela e o jogador deverá acertar com o dedo nele antes que suma. O game deve somar um ponto toda vez que o jogador acertar o personagem e subtrair toda vez que o jogador deixar escapar um personagem.

Até agora, os aplicativos tinham comportamento estático, ou seja, o estado atual não dependia do tempo de execução. Nesse aplicativo, usaremos o componente Temporizador, que tornará o comportamento do aplicativo dinâmico. Assim, mesmo que o usuário não dê nenhuma entrada, o app continuará sua execução. Além disso, veremos o componente SpriteImagem, uma figura que pode receber toques e agir como um botão, mas que pode se mover pela tela.

Componentes

1. Componente SPRITEIMAGEM (menu 'desenho e animação')

Diferente do componente Imagem, o componente SpriteImagem pode interagir com o usuário. Podemos pensar nele como um componente Botão cuja localização pode variar ao longo da execução do aplicativo. Muito provavelmente, os personagens dos aplicativos de games que você conhece são SpriteImagens. Por ser um personagem clicável, o SpriteImagem só pode ser posicionado dentro de um componente Pintura.

As propriedades do componente SpriteImagem são:

- **Ativo:** Indica se o componente inicia o aplicativo já ativo.
- **Direção:** Devolve a direção do Sprite em graus em relação ao eixo x positivo. Zero grau é para a direita da tela; 90 graus está voltada para cima de tela.
- **Altura:** Define a altura do componente.
- **Largura:** Define a largura do componente.
- **Intervalo:** O intervalo, em milissegundos, em que a posição do Sprite é atualizada. Por exemplo, se o intervalo é de 50 e a velocidade é de 10, então o Sprite se moverá 10 pixels a cada 50 milissegundos.
- **Imagem:** Define a imagem a ser usada como pano de fundo para o componente.
- **Rotação:** Se for selecionado, a imagem do Sprite gira para coincidir com a Direção atual do Sprite. Se false, a imagem do Sprite não gira quando houver alteração na Direção. O Sprite gira em torno do seu ponto central.
- **Velocidade:** A velocidade em pixels por Intervalo.
- **Visível:** Define se o componente está visível ao usuário.
- **X e Y:** Define a posição em termos de coordenadas X e Y do Sprite, obedecendo a orientação do App Inventor.
- **Z:** Define a camada em que o Sprite aparece em relação a outros Sprites, com camadas de número maior na frente das com números mais baixos.

2. Componente TEMPORIZADOR (menu 'sensores')

O Temporizador é um componente não-visível que fornece o instante no tempo usando o relógio interno do aparelho. Ele pode disparar o temporizador em intervalos regulares e executar cálculos de tempo, manipulações e conversões. Métodos para converter um instante de tempo para texto também estão disponíveis para padrões como DD/MM/AAAA, HH:MM:SS.

As propriedades do componente Temporizador são:

- **DisparosContínuos:** Se essa opção estiver selecionada, o Temporizador será acionado mesmo que o aplicativo esteja fechado.
- **Ativado:** Indica se o componente inicia o aplicativo já ativo.
- **Intervalo:** Indica o intervalo de tempo que o Temporizador irá marcar.

Programação – Modelagem IPO

Iremos agora fazer um jogo de Clique no Rato. Uma imagem de Rato irá aparecer em locais aleatórios da tela e o objetivo do jogador é clicar na imagem. Para esse aplicativo, iremos precisar dos seguintes componentes:

- 1 Pintura
- 1 SpriteImagem
- 1 Temporizador
- 1 Legenda

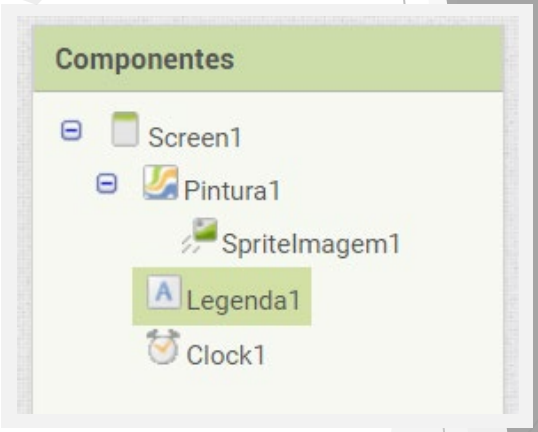


Figura 59: componentes

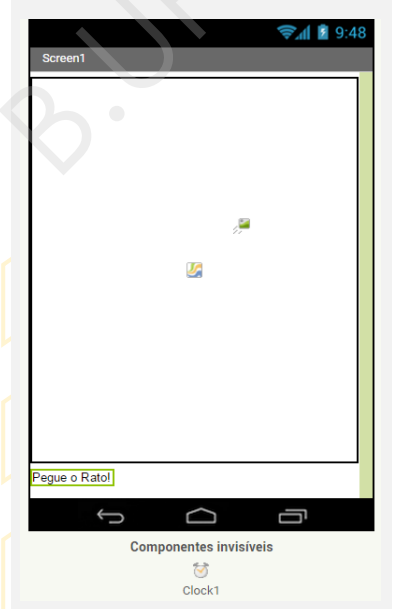


Figura 60: componentes na tela

Veja o exemplo ao lado e tente reproduzi-lo alterando as propriedades do seu aplicativo.

Vamos agora estudar as Entradas, Processamento e Saídas:

ENTRADAS	PROCESSAMENTO	SAÍDAS
1. Toque do usuário no SpriteImagem	1. A cada tempo do Temporizador, o SpriteImagem deverá ir para uma posição diferente da tela 2. Ao ser tocado, o SpriteImagem deverá avisar o programa que o aplicativo deve ser encerrado	1. A Legenda mudará de "Pegue o Rato!" para "PARABÉNS!" ao receber o toque do SPriteImagem.

Figura 61: programação IPO

A parte do processamento desse aplicativo tende a ser muito complexa. Por isso, apesar de termos apenas uma entrada, teremos mais de um gatilho, conforme veremos a seguir.

Programando nosso App

Antes de começar, vamos configurar nosso SpriteImagem. Para isso, precisamos subir uma imagem de um rato, que será nosso personagem chave do aplicativo, no menu de Mídia do App Inventor. Uma vez que tenhamos feito essa configuração, iremos programar o Gatilho de toque do Personagem, conforme abaixo:

Figura 62: programando o gatilho de toque do personagem

Note que o Gatilho pula da Entrada para a Saída. E o processamento, quem faz? Existem etapas do Processamento que acontecem independente das entradas e saídas do programa, e nesse aplicativo é isso o que acontece. A etapa que move o rato para diversas regiões é uma dessas etapas de processamento ocultas, cujo gatilho independe da ação externa. Por isso, usamos o Componente Temporizador. Veja o código abaixo:

Figura 63: programando o componente Temporizador

O que ele faz é, a cada disparo do Temporizador Clock1, mover o SpriteImagem para a posição X=0, Y=0, ou seja, o canto superior esquerdo da tela. Para que o game faça o personagem se mover aleatoriamente pela tela, precisamos pedir que, a cada disparo do Temporizador, o Sprite se mova para uma coordenada aleatória. Isso pode ser feito trocando os zeros pelo bloco de número aleatório.

Note que a coordenada X deveria variar entre 0 e a largura da tela, enquanto a coordenada Y deveria variar entre 0 e a altura da tela. Essas informações podem ser obtidas usando os atributos provenientes da tela. Entretanto, para nosso aplicativo faz mais sentido obter a altura e largura da TelaDePintura, para evitar que o Sprite apareça fora da área tocável.

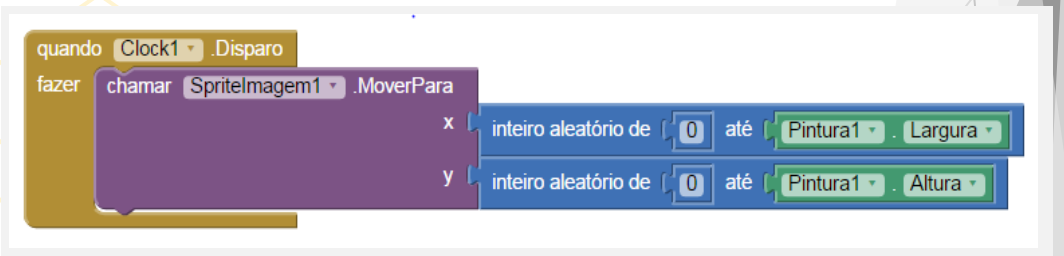


Figura 64: programando com bloco de número aleatório

Entretanto, note que, como a coordenada do Sprite é determinada pelo canto superior esquerdo, pode acontecer algo como o que é representado na imagem ao lado:

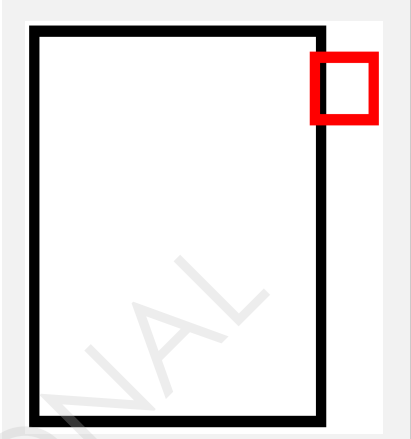


Figura 65: erro de tela

Se a Pintura1 for o retângulo preto e o Sprite for o quadrado vermelho, note que a representação indica que o Sprite está dentro da área do canvas. Portanto, essa é uma possibilidade real se deixarmos o sorteio aleatório de coordenadas como está. Precisamos então subtrair da coordenada máxima do Sprite o tamanho do próprio Sprite, a fim de garantir que ele esteja totalmente dentro do Canvas em todas as iterações do Temporizador. Assim, o código final do desse gatilho fica como o que vemos a seguir:

Figura 66: código do gatilho

Exercícios

Chegou a hora de praticar! Siga os passos abaixo e salve seu trabalho.

1. Crie um contador de pontos para seu game. Ou seja, ao tocar no Sprite, não termine o jogo, apenas some uma variável.

2. Faça a legenda permanecer com a palavra "parabéns" por apenas 2 segundos. Então esta legenda deve ser esvaziada. (Necessário um segundo temporizador).

3. Coloque na legenda o termo "GAME OVER", quando o usuário chegar a 20 pontos.

Desafio extra

1. Mude a velocidade de troca do Sprite a cada vez que o jogador acertar um toque no Sprite.

2. Mude a velocidade de troca do Sprite a cada vez que o jogador tocar na tela, mas não tocar no Sprite.

UNIDADE 6

CONEXÕES EXTERNAS E GEOLOCALIZAÇÃO

PROJETO 'MAPA TURÍSTICO'

PROJETO 'MAPA TURÍSTICO'

Contexto

Quando viajamos, por vezes deixamos de visitar alguma atração turística por não saber de sua existência ou por não saber como chegar nela. Para ajudar os turistas, nossa empresa pensou em um aplicativo que liste diversas atrações turísticas de uma cidade, e suas localizações. Faça esse aplicativo pensando nos pontos e atrações turísticas de sua cidade ou região!

Nem sempre os aplicativos terão atuação somente dentro do seu processamento. O grande poder dos aplicativos móveis está no fato de que eles podem se comunicar, via internet, com outras bases de dados ou ferramentas, enriquecendo as possibilidades do app. Nesse aplicativo, por exemplo, usaremos os dados do Google Maps para identificar as localizações dos pontos de interesse para os usuários.

Componentes

1. Componente INICIADORDEATIVIDADES (menu 'conectividade')

O IniciadorDeAtividades é um componente com o qual você pode iniciar qualquer aplicativo para Android, incluindo o Google Maps ou outro de seus próprios aplicativos. Trata-se de um componente de baixo nível, em que você precisará definir algumas propriedades com informações familiares a um programador Java Android SDK. Por isso, ao selecionar as propriedades tenha cuidado: se uma letra é maiúscula ou minúscula, é importante.

A depender da atividade a ser acionada, determinadas propriedades devem ser selecionadas. A seguir, listamos alguns dos aplicativos mais comuns e as respectivas propriedades.

Para abrir a câmera padrão do Android:

- Ação: android.intent.action.MAIN
- PacoteDaAtividade: com.android.camera
- ClasseDeAplicativo: com.android.camera.Camera

Para realizar uma busca na web, por exemplo, a palavra "BuildingUp":

- Ação: android.intent.action.WEB_SEARCH
- ChaveExtra: query
- ValorExtra: BuildingUp (para buscar outra palavra, substitua esse atributo) PacoteDeAtividade: com.google.android.providers.enhancedgooglesearch
- ClasseDaAtividade: com.google.android.providers.enhancedgooglesearch.Launcher

Para abrir uma página Web específica, por exemplo, a página da BuildingUp:

- Ação: android.intent.action.VIEW
- UriDeDados: http://www.buildingup.com.br

Programação – Modelagem IPO

Novamente, a modelagem IPO aqui é simples. Nosso aplicativo tem apenas uma entrada:

ENTRADAS	PROCESSAMENTO	SAÍDAS
1. Usuário abre o aplicativo	1. Aplicativo verifica lista de pontos turísticos e informações	1. Imprime o resultado na tela

Figura 67: programação IPO

Programando nosso App

1. Usando o Iniciador de Atividades para abrir o Google Maps

Primeiro, considere a URL *http://maps.google.com?q=SãoPaulo*. Quando você digita esta URL na barra de endereços de um navegador, ele mostra um mapa de São Paulo. O "?" É comum a muitas URLs; significa que um parâmetro está vindo em seguida.

Um parâmetro é a informação que o site precisa para processar a solicitação. Nesse caso, o nome do parâmetro é "q" (de *query,* que significa consulta em inglês) e seu valor é "SãoPaulo". Ele instrui o Google Maps que mapa deve ser exibido. Veja no exemplo abaixo como o aplicativo é montado. Note que alteramos o atributo Ação para indicar que nosso IniciadorDeAtividades será responsável por abrir uma URL.

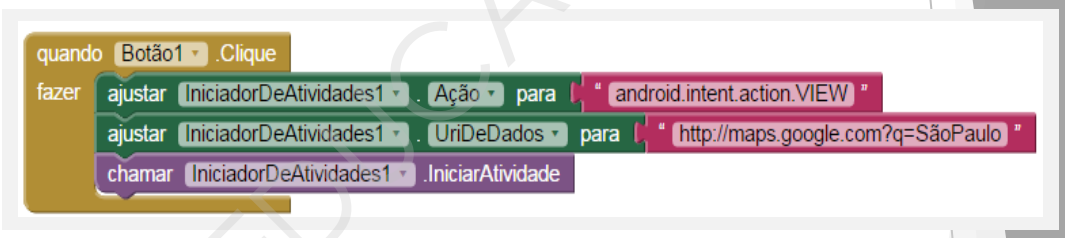


Figura 68: programação o IniciadorDeAtividades

Imagine agora que você deva escolher uma entre duas localidades, por exemplo dois locais de São Paulo. Você poderia criar 2 botões, cada um exibindo uma localidade para o usuário. Além disso, podemos realizar a mudança do atributo Ação para o momento me que a tela é inicializada, já que o comportamento é comum a todos os botões. Veja abaixo como ficaria essa construção:

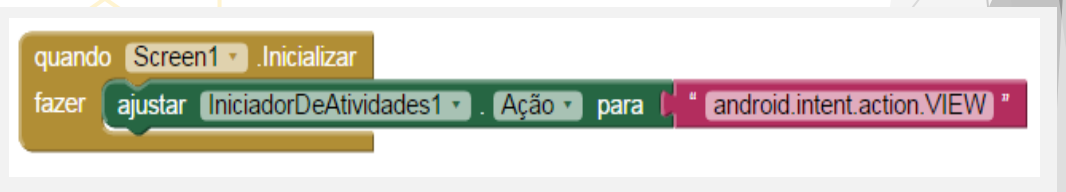


Figura 69: alternando localidades

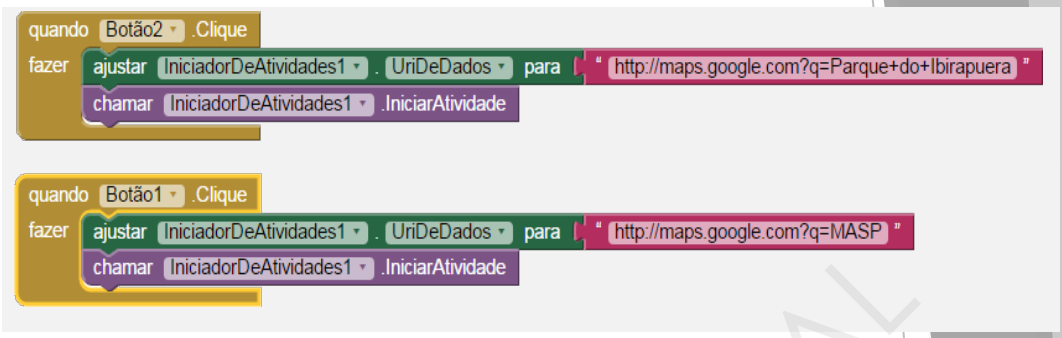


Figura 62: alternando localidades

A propriedade *UriDeDados* indica o endereço específico que deve ser aberto. Note que você pode (e na maior parte dos casos, deve) alterar dinamicamente o endereço. Por exemplo, vamos criar uma caixa de texto para que o usuário procure pelo mapa de sua cidade. O novo bloco ficaria assim:

Figura 70: criando caixa de texto

Agora o aplicativo faz a busca pela localidade descrita pelo usuário numa CaixaDeTexto.

Vamos agora criar uma lista com as duas localidades, usando o componente **ListaSuspensa (menu Interface de Usuário).** O funcionamento desse componente é muito semelhante ao EscolheLista que vimos em projetos anteriores. O bloco abaixo cria a lista de opções:

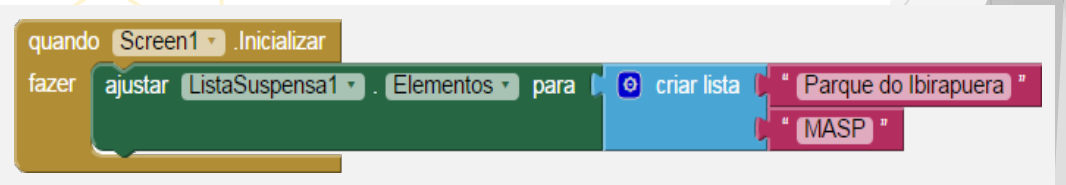


Figura 71: criando lista de opções

Ao selecionar uma opção, o aplicativo fará o que estiver dentro do bloco **DepoisDeSelecionar**. No nosso exemplo, abrirá o mapa da atração escolhida, como programado abaixo:

Figura 72: bloco DepoisDeSelecionar

2. Associando um texto para cada elemento

Imagine agora que queremos um texto descritivo para cada item. Por exemplo, queremos uma breve descrição do local escolhido. Para isso, podemos criar outra lista, em que os textos estarão guardados.

Note que os ***textos descritivos precisam estar na mesma ordem que seus respectivos locais***. Por exemplo, o texto relativo ao Parque do Ibirapuera precisa ser o primeiro, e o relativo ao MASP precisa ser o segundo. Isso porque as listas trabalham com índices, que indicam a posição na lista do elemento escolhido. Observe o exemplo abaixo:

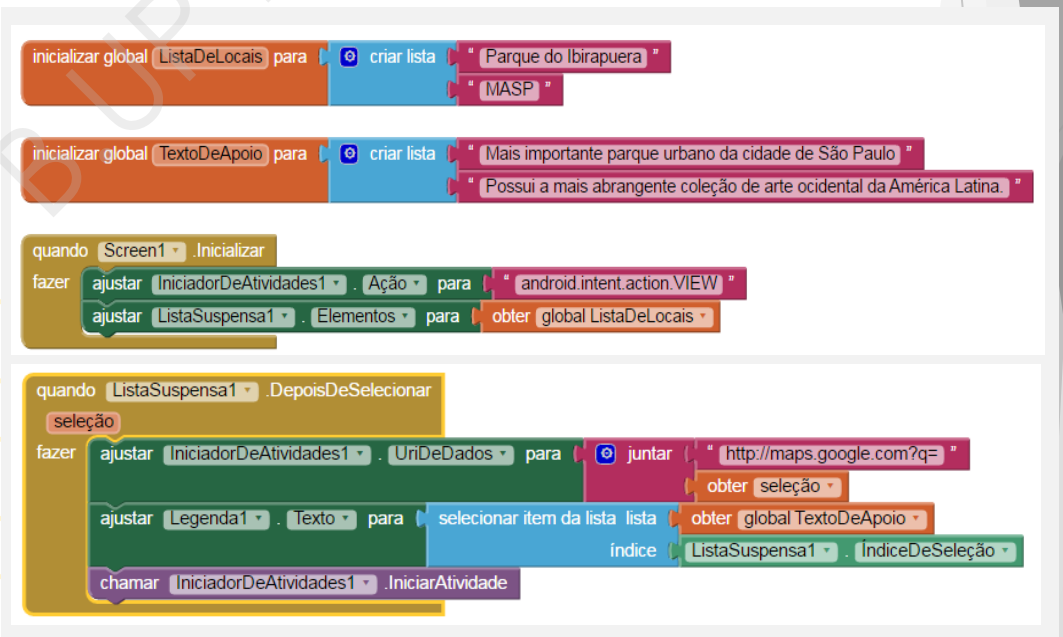


Figura 73: associando um texto para cada elemento

Note que quando o usuário seleciona o local, o app já abre o Google Maps e ele não tem tempo de ler o texto. Assim, alteraremos a implementação, criando um novo botão que abrirá o Google Maps, permitindo que o usuário leia o texto de apoio. Esse novo botão será habilitado somente quando o usuário escolher um local. Veja abaixo os blocos alterados:

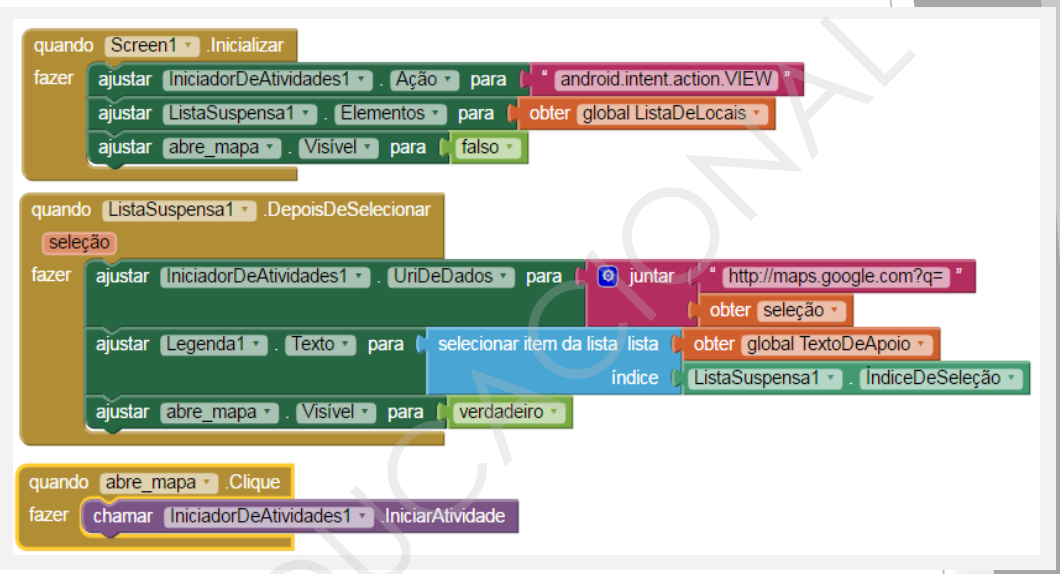


Figura 74: habilitando um botão para abrir o Google Maps

Nesse exemplo, criamos 2 variáveis que ao invés de armazenar apenas valores, armazenam uma lista inteira! Essa versatilidade nos possibilita diminuir a quantidade de código escrito, tornando o programa mais enxuto. Faça o teste, criando as variáveis e colocando-as no seu programa.

Exercícios

Chegou a hora de praticar! Siga os passos abaixo e salve seu trabalho.

1. Crie um botão que, ao ser clicado, direciona o usuário a uma foto do local escolhido. Essa foto pode ser um link da internet adicionado numa terceira lista. Lembre-se sempre de colocar os itens das listas em ordem, para que seu programa funcione como deveria!

2. Crie um botão que ao ser clicado, redireciona o usuário para a página oficial da atração turística.

Desafio extra

Deixe o aplicativo multi-cidade. Ao iniciar, o usuário deverá escolher a cidade desejada e só deverá exibir as atrações turísticas desta cidade.

UNIDADE 7

MECÂNICA E COLISÃO DE SPRITES, MATEMÁTICA

PROJETO 'GAME PONG'

PROJETO 'GAME PONG'

Contexto

Pong, lançado em 1972, é um jogo de esportes bidimensional que simula tênis de mesa. O jogador controla uma raquete movendo-a verticalmente em todo o lado esquerdo da tela, e compete contra um oponente controlado por outro jogador no lado oposto. Pontos são ganhos quando há uma falha em devolver a bola para o outro.

Esse estilo de jogo funcionava bem na época em que jogar video games era um evento entre amigos ou família. Hoje em dia, os games se adaptaram à uma nova realidade, em que uma pessoa pode querer jogar pelo celular ou tablet, enquanto espera numa fila de banco, por exemplo. Por isso, sua empresa decidiu investir nesse tema, e pediu para você recriar o Pong para dispositivos móveis.

Um dos conceitos centrais da produção de todo *game* são as colisões entre Sprites. Essa forma de interação, com os Sprites reagindo de acordo com o toque de um ou outro tipo, é a essência das mecânicas dos games. Apesar do App Inventor não ser uma ferramenta específica para criação de games, é possível criá-los e realizar a colisão entre os Sprites através de algumas adaptações na programação, que veremos a seguir.

Componentes

Para este projeto, não utilizaremos componentes novos, apenas os que já vimos em outros projetos.

Colisão entre *Sprites*

Para programar games é fundamental entender o conceito de colisão de Sprites. O App inventor não tem nativamente uma maneira de lidar com isso, por essa razão é necessário programar "na raça" essas opções. Assim, é necessário entender o conceito de Hitbox, ou caixa de colisão.

Hitbox é uma forma invisível, comumente utilizada em video games para detecção de colisão em tempo real. Por vezes, é um retângulo (em jogos 2D) ou um Ortoedro (em jogos 3D) e segue um ponto em um objeto visível (como um modelo ou Sprite), embora formas circulares e esféricas também sejam comuns. É comum objetos animados possuírem um conjunto de Hitboxes vinculados, cada um a uma parte em movimento, para garantir precisão durante o mesmo.

Hitboxes são usados para detectar colisões, como um personagem sendo acertado por um soco ou uma bala. Para utilizar as hitboxes, colocamos nossos personagens "dentro" de formas geométricas, como no exemplo da figura ao lado:

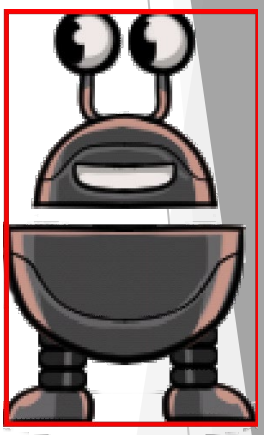
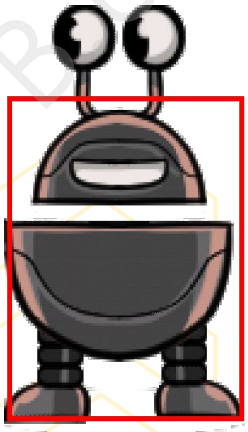


Figura 75: utilizando hitboxes



No exemplo, o robô é nosso personagem, e a forma vermelha é uma possível hitbox para esse Sprite. Note que a hitbox ultrapassa os limites da imagem. Podemos escolher outras hitboxes mais precisas, mas quanto mais arestas tiver a forma escolhida, mais trabalhosa será a programação. Pode-se, então, escolher um meio termo entre a precisão e a complexidade computacional desejada, como no exemplo ao lado.

Figura 76: utilizando hitboxes

Note que a hitbox tornou-se mais próxima ao desenho original, sem que para isso tivéssemos que aumentar sua complexidade. E por que usamos hitboxes? Para definir se suas imagens colidiram, precisamos descobrir se algum ponto de uma das imagens está dentro do espaço formado pela outra imagem. Por exemplo, na imagem ao lado, sabemos que os robôs colidiram, pois suas hitboxes coincidem na área pintada de amarelo.

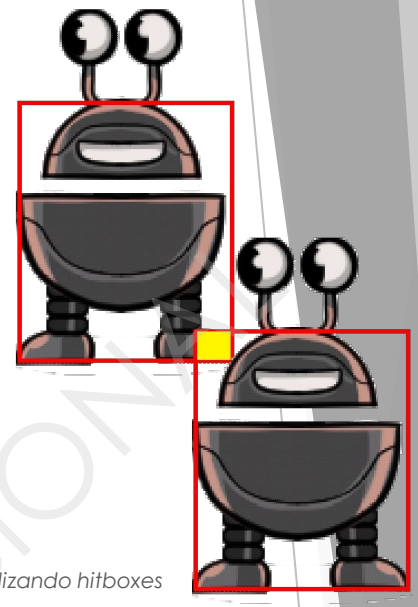


Figura 77: utilizando hitboxes

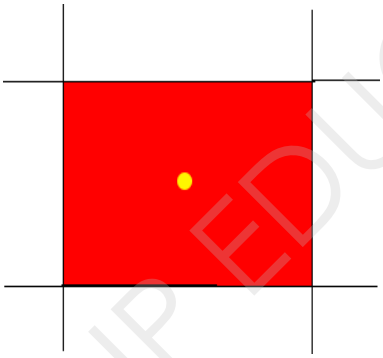


Figura 78: coordenadas X e Y

Para identificar se um ponto está dentro de um retângulo é fácil: basta saber se a coordenada X do ponto em questão está entre as coordenadas X dos vértices do retângulo, e se a coordenada Y do ponto está entre as coordenadas Y do retângulo, como mostra a imagem ao lado.

Para polígonos mais complexos, determinar se um ponto está dentro dele é tarefa complicada, sendo tema de diversos trabalhos acadêmicos de universidades mundo afora. Por isso, o App Inventor escolheu abrigar seus Sprites em retângulos, e é isso que iremos usar em nosso projeto Pong.

Programando nosso App

A interface do usuário para o nosso aplicativo Pong consiste em vários componentes que você usou antes:

- uma legenda para a pontuação;
- dois botões (iniciar e reiniciar);
- um ArranjoHorizontal que contém esses três componentes
- uma Pintura;
- um SpriteImagem para representar a raquete;
- um componente Bola, que possui as mesmas características do SpriteImagem;

Na verdade, o componente Bola é um SpriteImagem, e foi separado pelo App inventor para facilitar o aprendizado inicial. Veja abaixo o esquema de como ficará o App e seus componentes:

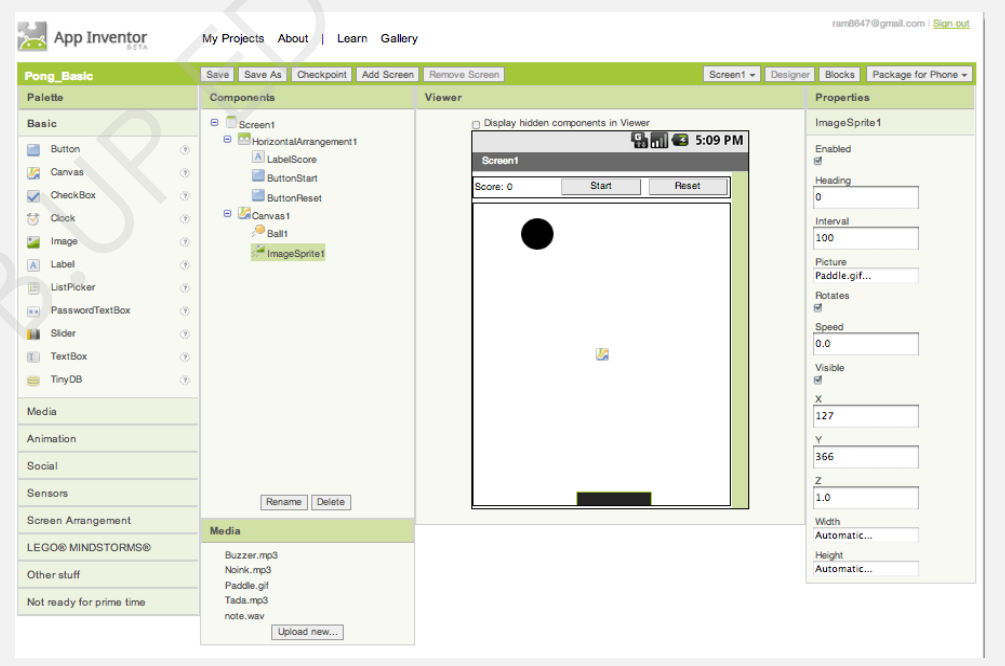


Figura 79: esquematização do app e seus componentes

Observe as propriedades diferentes do ImageSprite. Você verá que a bola tem as mesmas propriedades, exceto para a propriedade de imagem. O valor desta propriedade é o nome de um arquivo de imagem ("Paddle.gif"), que você deverá subir para a plataforma para representar a raquete. Essa imagem deve ser um retângulo.

A parte principal da nossa tarefa de programação para este aplicativo será o controle dos valores dessas propriedades.

O Canvas contém dois componentes, um Ball e um ImageSprite, ambos componentes de animação Bolas e ImageSprites. Ambos os componentes podem responder a eventos de toque e colisões com outras bolas ou sprites ou com as bordas da tela. A principal diferença é que um ImageSprite pode ter uma imagem associada a ele, neste caso um retângulo preto, que representa o "Paddle".

Canvas

Já usamos o Canvas (ou Pintura) em outros projetos, mas vamos nos aprofundar em seu conteúdo. O App Inventor atribui números às 4 arestas da tela, conforme mostrado na imagem abaixo. Por exemplo, para detectar quando a bola atingiu a borda inferior usamos a condição edge = -1.

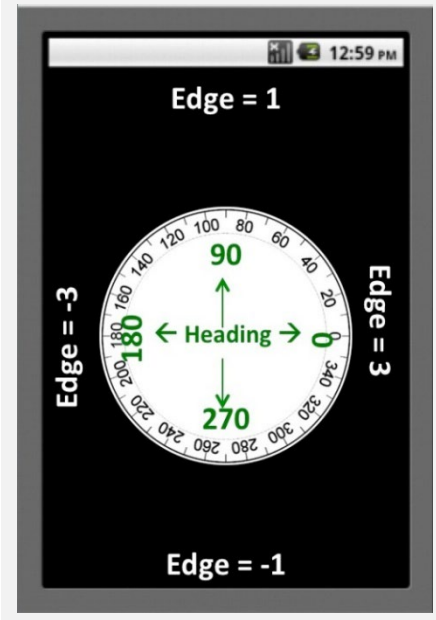


Figura 80: canvas e direção

Além disso, observe como a propriedade *Direção* da bola e do *Sprite* é definida: uma Direção de 90 graus significa que a bola ou o *Sprite* está voltado para o topo da tela. Uma direção 0 significa que a bola está em direção à borda direita. Note que uma reflexão numa superfície plana pode ser feita simplesmente calculando o valor de 360 – Direção.

Veja exemplos na tabela abaixo:

Direção Atual	Direção Nova
90	360 - 90 = 270
240	360 - 240 = 120

Tabela 2: direções

Vamos usar esta expressão para fazer a bola rebater ou ricochetear na direção oposta quando ela bate no paddle.

Controlando o paddle

Para controlar o 'arrastar' do Paddle, usamos o bloco ImageSprite1.Dragged. Como o Paddle só pode mover-se horizontalmente, sua coordenada Y sempre será a mesma. No entanto, sua coordenada X será alterada para currentX, que representa o atual valor da coordenada X do toque na tela. Assim, usamos um bloco get para obter a coordenada currentX do paddle, que é uma das entradas do bloco Dragged com você vê abaixo:

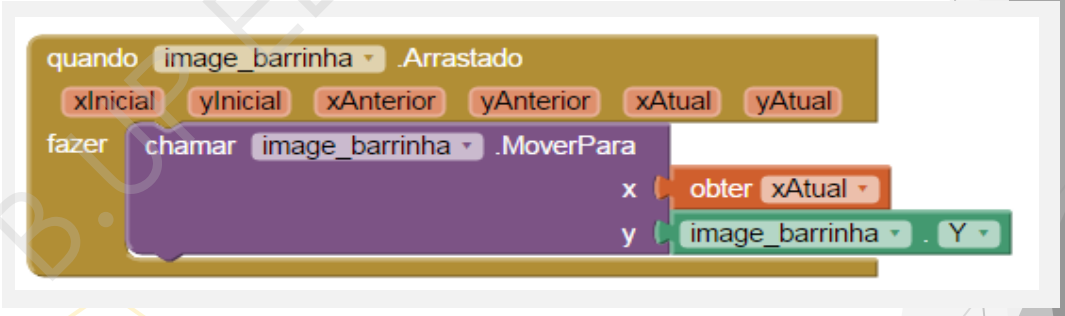


Figura 81: bloco para controle do paddle

Quicando na borda

Para fazer a bola quicar nas bordas da tela, basta usar o código abaixo. Isso fará a bola rebater em todas as bordas. Observe como usamos um bloco get para dar ao procedimento o valor da entrada EdgeReached.

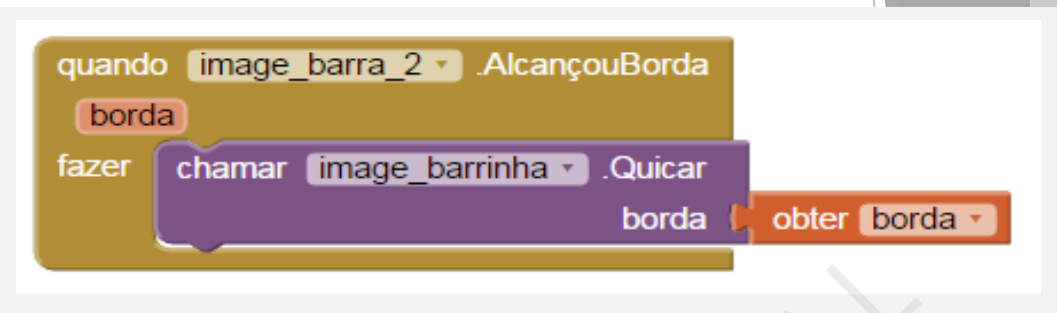


Figura 82: bloco para quicar na borda

Colisão com o paddle

Finalmente, temos que gerenciar as colisões entre a bola e o paddle. Tudo o que acontece aqui é a bola quicar no paddle, indo na direção oposta. Aqui é onde usamos a expressão (360-Direção) para inverter a direção da bola:

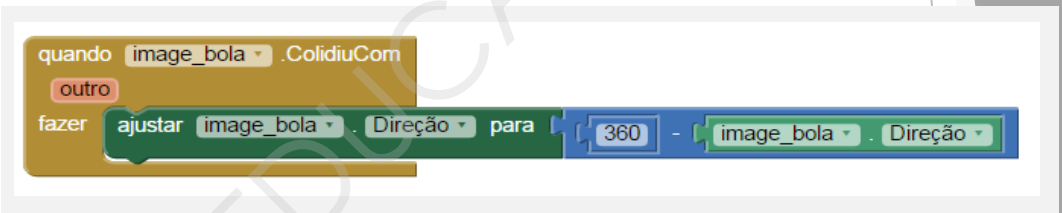


Figura 83: bloco para colidir com o paddle

Resta-nos agora definir os parâmetros iniciais do game. Por exemplo, a velocidade inicial da bola, a direção inicial dela e a posição inicial do Paddle.

Figura 84: bloco parâmetros iniciais

Note que posicionamos nossa barrinha na metade do Canvas. Além disso, definimos uma direção aleatória para a bola iniciar. Finalmente, observe que a velocidade da bola será fixa, já que nunca mexemos na propriedade Velocidade dela.

Finalmente, temos os botões de Start e Restart. Quais são as condições para iniciar o jogo? Ou seja, qual a diferença entre o jogo antes e depois de iniciar? Uma das condições é que a bola e o Paddle devem se manter parados, somente ficando ativos quando o botão for pressionado. Assim, alteraremos a função Inicializar do Screen como você vê abaixo:

Figura 85: alterando a função 'inicializar' do screen

Já o botão de reinicializar é semelhante ao de start, mas com uma adição: a posição da bola deve voltar ao meio da tela, e outra direção deve ser escolhida. Como você faria essa alteração?

Exercícios

Chegou a hora de praticar! Siga os passos abaixo e salve seu trabalho.

1. Faça o botão de restart.
2. Encerre o game quando a bola tocar a borda de baixo.

Desafio extra

1. Crie um contador de rebatidas da bola na raquete. Quando esse contador chegar a 10, dobre a velocidade da bola.
2. Usando o contador de rebatidas, crie um código que aumenta a velocidade da bola em 10% a cada 10 rebatidas.

UNIDADE 8

PROGRAMAÇÃO, ÁLGEBRA E GEOMETRIA

PROJETO 'MINI GOLF'

PROJETO 'MINI GOLF'

Contexto

Seu game anterior foi um sucesso, e por isso decidiu-se por continuar na área. Agora, seus chefes pediram para você criar um game de MiniGolf, no qual o usuário deverá atingir uma bola com um "peteleco" e ela irá deslizar pela tela, batendo em obstáculos até chegar ao final.

Este Mini Golf App demonstra como usar os gestos Fling, TouchUp e TouchDown para Sprites. Note que esses gatilhos de eventos também estão disponíveis para o Canvas.

Para jogar este Mini Golf App, o jogador posiciona sua bola dentro dos limites do campo, e depois joga a bola em direção ao buraco. A bola deve quicar no obstáculo retangular e nos lados do percurso. Para cada lançamento da bola, a contagem aumenta em um. A pontuação total é o número de lançamentos necessários para completar todo o curso.

Programando nosso App

Construiremos este aplicativo em etapas, adicionando um pouco do jogo de cada vez.

Inicie uma sessão no App Inventor e comece um novo projeto. Nomeie-o "MiniGolf". Quando a janela de Designer é aberta, note que o App Inventor chama automaticamente a tela "Screen1", mas você pode definir o título da tela, que aparecerá na barra superior do aplicativo. Pense em um título relacionado ao Mini Golf e digite-o no painel de Propriedades no lado direito do Designer.

Nas propriedades de tela (mostradas no painel do lado direito), desmarque a caixa de seleção "rolável" para que a tela não seja exibida quando o aplicativo estiver sendo executado. Telas que estão definidas para rolar não têm uma altura. Vamos precisar da nossa tela para ter uma altura definida, a fim de configurar o campo de golfe corretamente.

Para esse game, iremos usar os seguintes componentes:

- 1 pintura
- 2 bolas: uma representando a Bola (GolfBall) e outro representando o Buraco (Hole)
- 1 spriteimage (Obstáculo)
- 1 temporizador

Figura 86: configurando o campo de golf

Note que o obstáculo ainda não foi adicionado a esse jogo, apesar de já ter um Componente dedicado para ele (um SpriteImagem).

Programando o comportamento da bola

Primeiro, use o manipulador de eventos GolfBall.arremessado para mover a bola de golfe quando ela é arremessada. Observe como esse gatilho recebe 6 argumentos diferentes:

1. X - a posição x do dedo do usuário
2. Y - a posição y do dedo do usuário
3. Velocidade - do gesto de arremesso do usuário
4. Posição - a direção (em graus) do gesto de arremesso
5. Xvel - a velocidade na direção X do arremesso do usuário
6. Yvel - a velocidade na direção Y do arremesso do usuário

Essencialmente você quer ajustar a velocidade e a direção do GolfBall para combinar a velocidade e a direção do gesto do arremesso do jogador. Você pode querer aumentar a velocidade um pouco, porque a velocidade do arremesso é um pouco mais lenta do que como uma bola de golfe iria se mover. Você pode jogar com este "fator de escala" para fazer a bola mais ou menos sensível a um arremesso. Quanto maior o valor multiplicado pelo atributo Velocidade, maior a sensibilidade do arremesso.

Veja o código abaixo. O número 2 representa uma escala de sensibilidade da velocidade. Ajuste esse valor para ficar da maneira que lhe agradar mais.

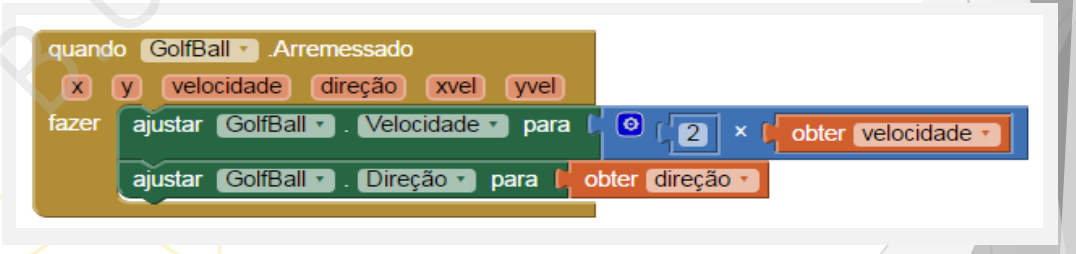


Figura 87: programando o comportamento da bola

Quicar quando tocar na borda

Para fazer a bola quicar nas bordas, faremos um bloco semelhante ao que fizemos no projeto Pong, usando o gatilho 'AlcançouBorda'.

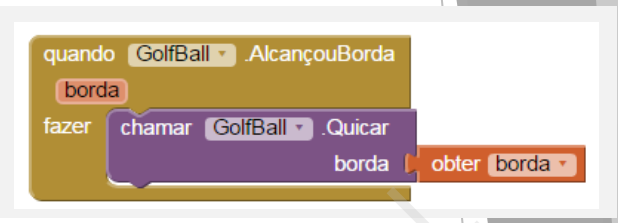


Figura 88: programando a bola para quicar nas bordas

Programando o comportamento do relógio

Use o evento timer para diminuir a velocidade da bola para que não quique para sempre. Cada vez que o relógio dispara, ele irá reduzir a velocidade da bola ligeiramente. Observe que se a bola não está se movendo, então esses blocos não farão nada. Se não usássemos esse bloco, então a bola iria apenas saltar para sempre (faça o teste!).

Figura 89: programando o comportamento do relógio

Programando um procedimento chamado 'novo buraco'

Este procedimento será chamado quando um buraco é marcado e a bola tem de ser colocada de volta no ponto de partida. Observe que o bloco Hole.MoveTo define o buraco em um novo local aleatório para a próxima execução.

Figura 90: programando um 'novo buraco'

Note que para localizar o buraco, usamos um expediente parecido com o do projeto 'teste de reflexos', para nos assegurarmos que o buraco não apareça fora da tela. Entretanto, no caso do Componente bola, a referência de coordenada do componente é o centro da imagem, por isso usamos como medida o seu raio.

Finalmente, iremos programar o comportamento que indica que o game terminou, ou seja, que o buraco e a GolfBall se encontraram:

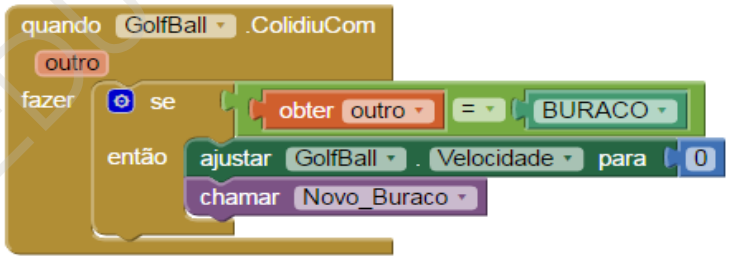


Figura 91: programando o fim do game

Para programar o obstáculo, iremos adicionar outra condição para a GolfBall. Para fazer a reflexão no obstáculo, usaremos a mesma equação que foi feita no projeto Pong. Veja abaixo:

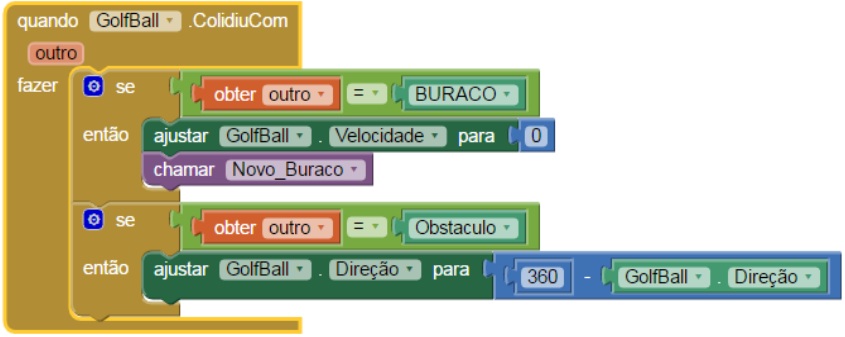


Figura 92: programando o obstáculo

Exercícios

Chegou a hora de praticar! Siga os passos abaixo e salve seu trabalho.

1. Crie um contador de jogadas, e se o jogador passar de 4 jogadas, termine o jogo.

2. Adicione uma legenda no seu game com a inscrição "Hole in One!", que deverá aparecer sempre que o jogador acertar o buraco com apenas uma jogada.

Desafio extra

Faça um obstáculo "de borracha", ou seja, quando a GolfBall rebater nele, além de alterar a direção, ele deverá diminuir a velocidade da bola pela metade.

UNIDADE 9

SENSOR DE LOCALIZAÇÃO E BANCO DE DADOS

PROJETO 'FUSION TABLES'

PROJETO 'FUSION TABLES'

Contexto

Quantas vezes você esteve passeando por uma cidade nova e viu um lugar legal para visitar, mas não podia parar naquela hora? E depois, não conseguiu mais achar aquele lugar, e acabou não fazendo a visita? Para solucionar esse problema, faça um aplicativo que o usuário pode marcar um local com o GPS e, depois, pode voltar a esse local, obtendo as direções para chegar nele pelo mapa do smartphone!

Neste aplicativo, iremos utilizar novamente o IniciadorDeAtividades, mas dessa vez ele será responsável por lançar o aplicativo de mapas do celular. Além disso, iremos aprender a usar o sensor de Localização. Note que esse sensor somente funcionará se seu dispositivo possuir um localizador, ou seja, se ele for capaz de emitir e receber sinais de GPS. Finalmente, usaremos o componente TinyDB, que simula um pequeno banco de dados no smartphone.

Componentes

1. Componente SENSORDELOCALIZAÇÃO (do menu 'sensores')

Componente não visível que fornece informações de localização, incluindo longitude, latitude, altitude (se suportado pelo dispositivo), velocidade (se suportado pelo dispositivo) e endereço. Isso também pode executar "geocodificação", convertendo um determinado endereço (não necessariamente o atual) para uma latitude (com o método LatitudeFromAddress) e uma longitude (com o método LongitudeFromAddress).

As propriedades do componente SENSORDELOCALIZAÇÃO são:

- **IntervaloDeDistância:** Determina o intervalo mínimo de distância, em metros, que o sensor tentará usar para enviar atualizações de local. Por exemplo, se estiver definido como 10, o sensor disparará um evento LocalizaçãoAlterada somente após 10 metros terem sido atravessados. No entanto, o sensor não garante que uma atualização será recebida exatamente no intervalo de distância. Pode demorar mais de 10 metros para disparar um evento, por exemplo;
- **Ativado:** Indica se o componente inicia o aplicativo já ativo;
- **IntervaloDeTempo:** Determina o intervalo de tempo mínimo, em milissegundos, que o sensor tentará usar para enviar atualizações de local. No entanto, as atualizações de local só serão recebidas quando a localização do telefone realmente mudar e o uso do intervalo de tempo especificado não é garantido. Por exemplo, se 1000 for usado como o intervalo de tempo, atualizações de local nunca serão disparadas antes de 1000m, mas eles podem ser acionados em qualquer momento após.

1. Componente TINYDB (do menu 'armazenamento')

O TinyDB é um componente não visível, sem propriedades, que armazena dados para um aplicativo. Apps criados com o App Inventor são inicializados cada vez que são executadas. Isso significa que se um aplicativo define o valor de uma variável e o usuário fecha o aplicativo, o valor dessa variável não será lembrado na próxima vez que ele for executado.

Em contrapartida, o TinyDB é um armazenamento de dados persistente, ou seja, os dados armazenados em um TinyDB estarão disponíveis sempre que o aplicativo for executado. Um exemplo pode ser um jogo que salva a pontuação mais alta e a recupera cada vez que o jogo é jogado.

Os itens de dados são strings armazenados em tags. Para armazenar um tem de dados, especifique a tag na qual ele deve ser armazenado. Posteriormente, você pode recuperar os dados que foram armazenados em uma determinada tag.

Cada aplicativo tem seu próprio armazenamento de dados. Há apenas um armazenamento de dados por aplicativo. Mesmo se você tiver vários componentes do TinyDB, eles usarão o mesmo armazenamento de dados. Para obter o efeito de lojas separadas, use chaves diferentes. Você não pode usar o TinyDB para passar dados entre dois aplicativos diferentes no telefone, embora você possa usar o TinyDB para compartilhar dados entre as diferentes telas de um aplicativo com várias telas.

Quando você está desenvolvendo aplicativos usando o AI Companion, todas as aplicações que utilizam esse Companion irão compartilhar o mesmo TinyDB. Essa partilha desaparecerá uma vez que os aplicativos são empacotados e instalados no telefone. Durante o desenvolvimento, você deve ter cuidado para limpar os dados do aplicativo AI Companion cada vez que começar a trabalhar em um novo aplicativo.

Propriedades do componente INICIADORDEATIVIDADES

Como vimos anteriormente, os atributos desse componente devem ser inicializados de acordo com a funcionalidade que irão chamar. Para chamar os apps de mapa, utilize as propriedades abaixo. As propriedades que não aparecerem na lista devem ser deixadas em branco. Lembre-se de colocar as propriedades exatamente como estão escritas abaixo:

- Ação: android.intent.action.VIEW
- ClasseDeAtividade:
  com.google.android.maps.MapsActivity
- PacoteDeAtividade: com.google.android.apps.maps

Programando nosso App

Para fazer nosso aplicativo, iremos primeiro construir sua interface com o usuário. Usaremos 2 botões, um para memorizar uma localização e outra para dar as direções partindo da localização atual para a localização memorizada. Além disso, colocaremos 2 legendas para indicar o endereço e a geolocalização do local memorizado.

Outros componentes de layout podem ser adicionados caso você deseje, mas para esse capítulo trabalharemos com os componentes já citados, como você vê abaixo:

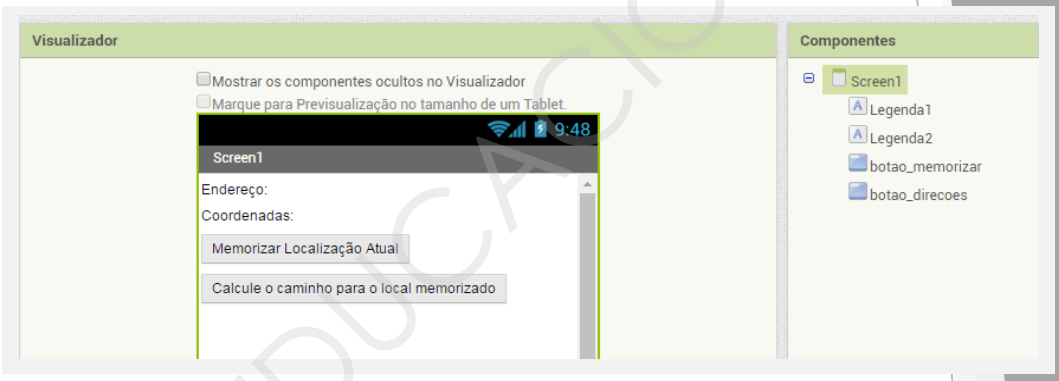


Figura 93: programando o aplicativo

Além disso, teremos os seguintes componentes invisíveis: SensorDeLocalização e TinyDB (menu armazenamento). Também teremos um IniciadorDeAtividades (menu Conectividade).

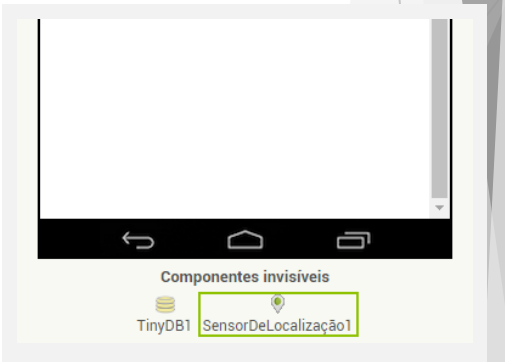


Figura 94: componentes invisíveis

Ao pressionar o botão de memorizar, o aplicativo deve armazenar a localização atual do aparelho, baseando-se nas informações obtidas pelo SensorDeLocalização. Note que, para identificar unicamente um local num mapa, basta que tenhamos suas coordenadas, a Latitude e a Longitude do local. Para isso, utilizaremos duas variáveis para armazenar as coordenadas:

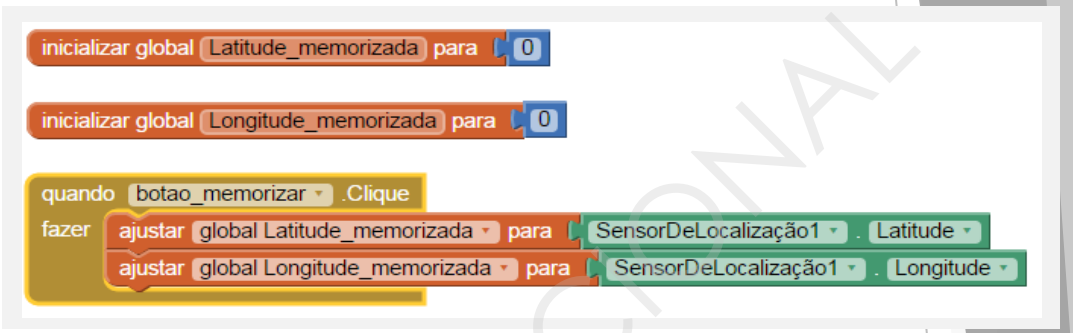


Figura 95: variáveis para armazenar coordenadas

O SensorDeLocalização armazena, seguindo as propriedades IntervaloDeDistância e IntervaloDeTempo, constantemente a localização atual. Por isso, ao utilizar os blocos verdes, obtemos a coordenada atual. Podemos exibir na tela, usando as Legendas, as coordenadas também. Veja abaixo uma sugestão de como fazer alteração no componente:

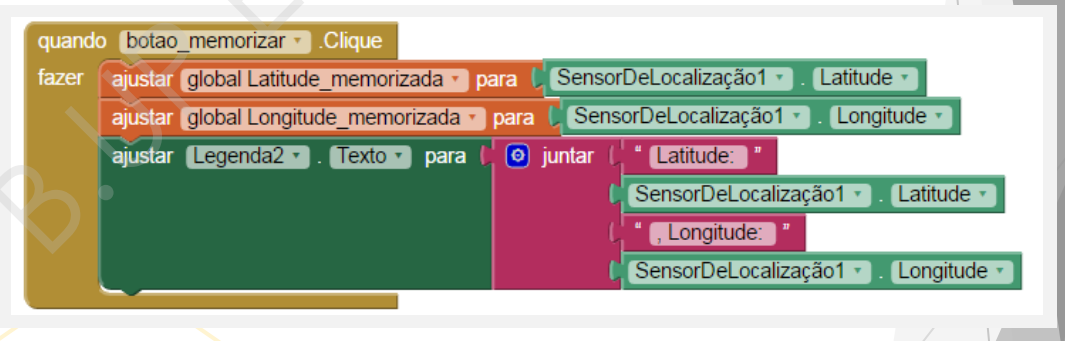


Figura 96: usando legendas para exibir coordenadas na tela

Além disso, podemos já alterar para mostrar o endereço do local, baseado nas coordenadas. Essa informação fica armazenada no atributo Endereço do Sensor de Localização.

Finalmente, devemos armazenar as informações sobre o local no Banco de Dados do TinyDB. Para isso, marcaremos com uma "etiqueta" cada informação, para que consigamos recuperá-las quando necessário. Veja a seguir como fica o bloco completo:

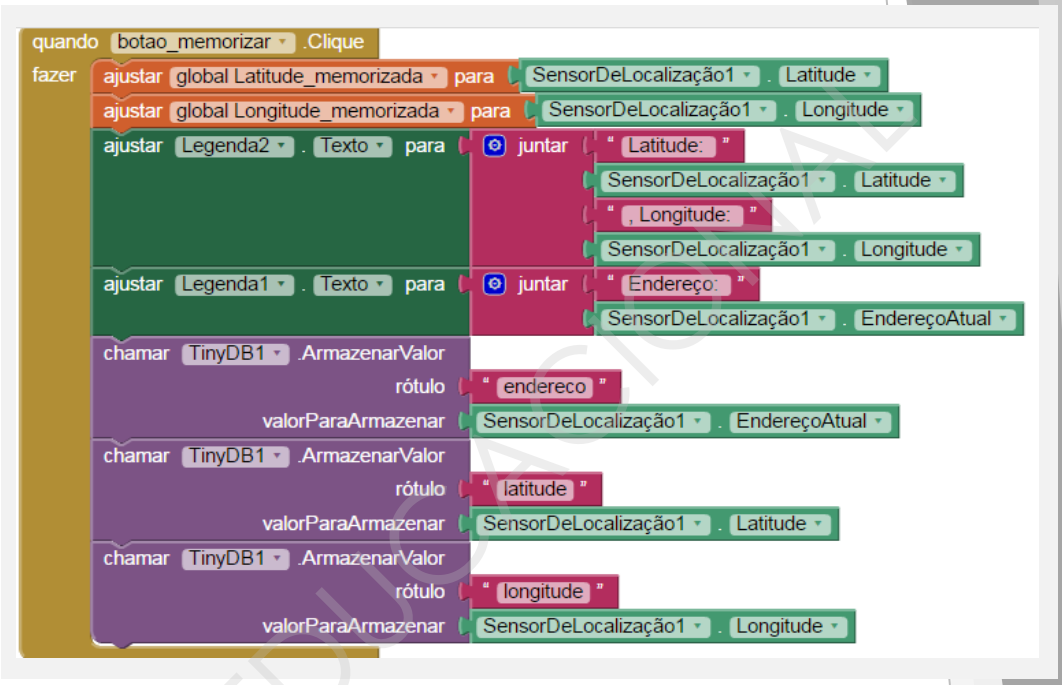


Figura 97: bloco de programação completo

Faremos agora o botão para obter as direções. Para isso, usaremos uma facilidade do Google Maps. Ele permite que informemos na própria URL (o endereço web do site) os locais de partida e de chegada, e ele já nos informa o melhor caminho entre os dois locais. A construção da URL é simples, e pode ser entendida com a construção da string que representará o endereço que o Google Maps deverá receber:

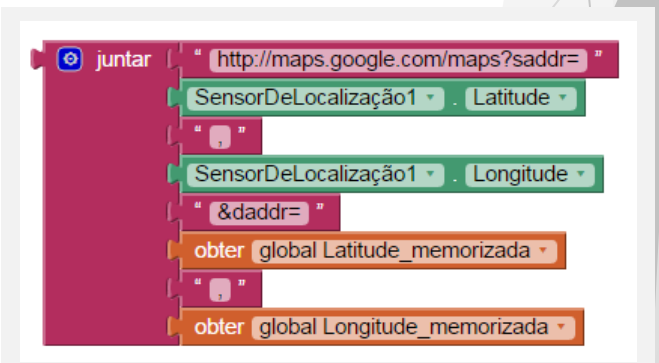


Figura 98: construindo uma string

Antes da palavra "saddr" deverão vir as coordenadas do local de início do trajeto, e após a palavra "daddr" deverão vir as coordenadas do destino (no nosso caso, o endereço memorizado).

Uma vez montada a URL, basta utilizar um componente IniciadorDeAtividade (adicione-o ao projeto e inicialize com as propriedades descritas anteriormente), como você vê abaixo:

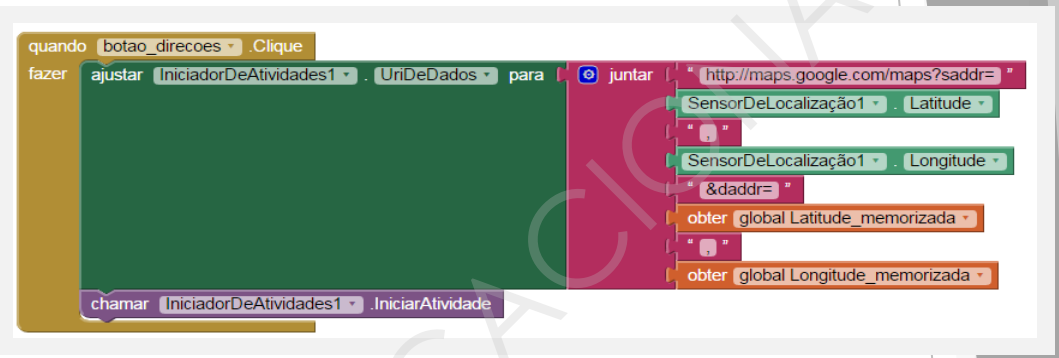


Figura 99: utilizando o componente IniciadorDeAtividades

Finalmente, precisamos inicializar a tela para que ela pegue os dados memorizados no TinyDB. Isso pode ser feito com os códigos abaixo. Note que os rótulos devem ser os mesmos do que já foi memorizado anteriormente.

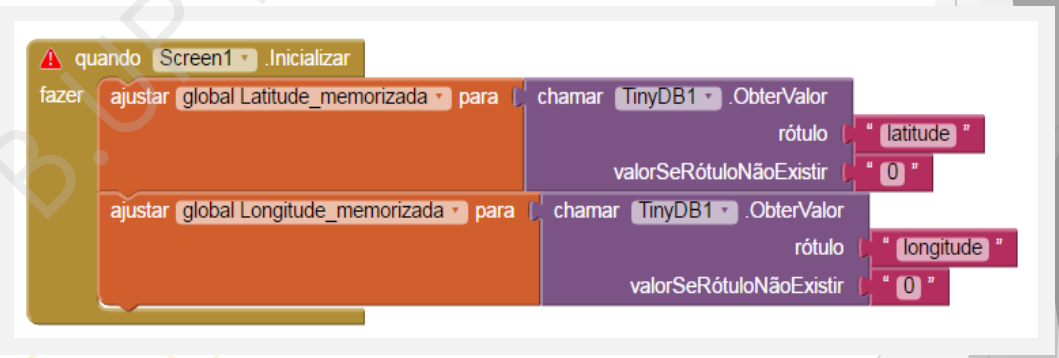


Figura 100: iniciando a tela para coletar dados do TinyDB

Exercícios

Chegou a hora de praticar! Siga os passos abaixo e salve seu trabalho.

Adicione um botão de Compartilhamento em seu app, usando o componente Compartilhamento (menu Social). Esse botão, ao ser clicado, deverá enviar uma mensagem contendo a posição memorizada pelo aplicativo.

Desafio extra

Modifique seu app com uma caixa de texto, onde o usuário pode escrever um endereço ou um local famoso. Seu app deverá memorizar o local obtido pela caixa de texto, usando os procedimentos "LatitudeDoEndereço" e "LongitudeDoEndereço" para obter as coordenadas desejadas.

UNIDADE 10

PROGRAMAÇÃO DE MÚLTIPLOS FATORES

PROJETO 'QUIZ ILUSTRADO'

PROJETO "QUIZ ILUSTRADO"

Neste projeto, vamos criar um app de perguntas e respostas com imagens ilustrativas. Em nosso exemplo, o tema será História do Futebol e terá apenas 3 perguntas, mas é claro que você pode acrescentar muitas outras questões.

O layout básico do aplicativo está ilustrado na imagem abaixo, bem como os seus respectivos componentes.

- 1 Image;
- 2 HorizontalArrangements;
- 3 Labels;
- 1 TextBox;
- 2 Buttons.

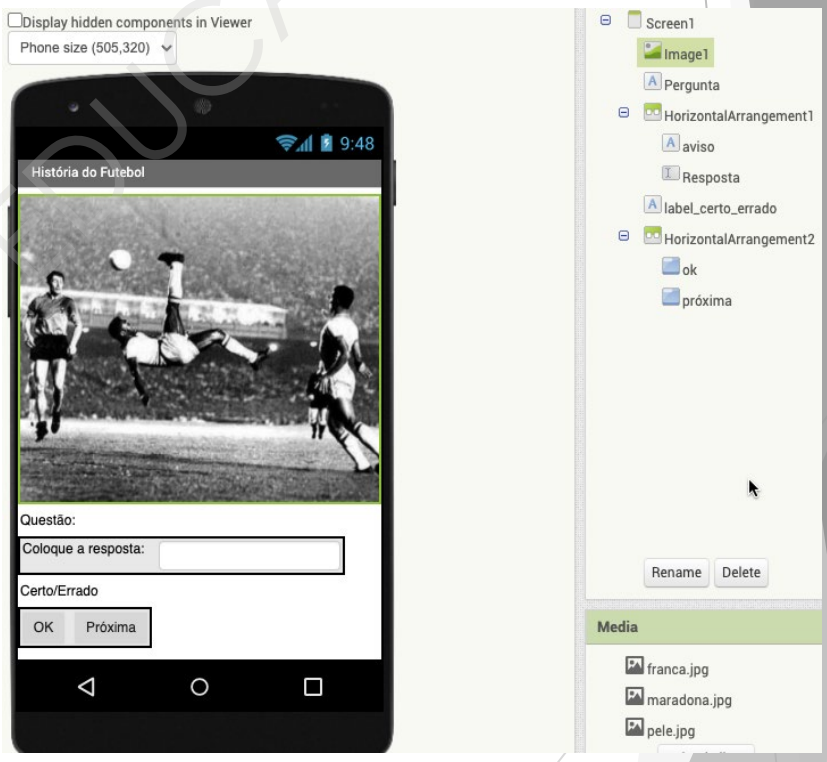


Figura 101: layout do app

Além destes componentes, são necessárias a lista de perguntas, a lista de respostas e as respectivas imagens. É importante que as perguntas sejam enunciadas de tal forma que haja apenas uma resposta verdadeira e ela seja apenas uma ou duas palavras.

A pergunta não precisa, necessariamente, se referir à sua imagem, que pode ser apenas uma ilustração da pergunta, mas é importante que cada pergunta tenha a sua própria imagem. A ordem da lista de perguntas deve corresponder à lista de respostas, ou seja, o primeiro item da lista de perguntas deve ser respondida pelo primeiro item da lista de respostas, a segunda pergunta deve ser respondida pela segunda resposta e assim por diante.

É interessante, também, que os nomes dos arquivos de imagem correspondam às respostas. No exemplo que veremos, o nome da imagem do Pelé é "pele.jpg", porque a resposta a esta pergunta é "Pelé". Os nomes dos arquivos de imagens não podem conter caracteres especiais, ou seja, deve se iniciar por uma letra e ser composta por um conjunto formado por letras (não acentuadas) ou números.

1. Componente IMAGE (do menu 'interface do usuário')

Componente para exibição de imagens e animações básicas. A imagem a ser exibida e outros aspectos da aparência da imagem podem ser especificados no *Designer* ou no Editor de blocos.

Propriedades do componente IMAGE

- **Animation:** Esta é uma forma limitada de animação que pode anexar um pequeno número de tipos de movimento às imagens. Os movimentos permitidos são *ScrollRightSlow, ScrollRight, ScrollRightFast, ScrollLeftSlow, ScrollLeft, ScrollLeftFast, Stop.*

- **Clickable**: Especifica se a imagem deve ser clicável ou não.
- **Height**: Especifica a altura da Image, medida em pixels.
- **HeightPercent**: Especifica a altura da Image, como uma percentagem da altura da Screen.
- **Picture**: Especifica o caminho do arquivo da Image.
- **RotationAngle**: O ângulo em que a Image está rotacionada. Essa rotação não aparece na tela do designer, apenas no dispositivo.
- **ScalePictureToFit**: Especifica se a imagem deve ser redimensionada para o tamanho do Viewer (não tem qualquer efeito no dispositivo).
- **Scaling**: Esta propriedade determina como a Image é dimensionada de acordo com a altura ou largura da imagem. Escala proporcionalmente (0) preserva a proporção da imagem. Dimensionar para ajustar (1) corresponde à área da imagem, mesmo se a proporção for alterada.
- **Visible**: Especifica se Image deve ser visível na tela. O valor é true se Image estiver sendo exibida e false se estiver oculta.
- **Width**: Especifica a largura horizontal do Image, medida em pixels.
- **WidthPercent**: Especifica a largura horizontal de Image como uma percentagem da largura da Screen.

Repare que nem todas as propriedades citadas aparecem no *Designer* do *App Inventor*, como *Animation* e *Scaling*, mas todas elas podem ser alteradas via programação dos blocos.

Vamos agora à construção do algoritmo! Os comportamentos a seguir são os esperados:

1. Aplicativo exibe uma imagem e uma pergunta associada a ela.
2. Quando usuário escrever a resposta correta a esta pergunta na *TextBox* e pressionar *Button* Ok, então o texto da *Label* "label_certo_errado" é alterado para "Certo", senão é alterado para "Errado".
3. Quando usuário pressiona o *Button* "próxima", então a *Label* "label_certo_errado" e o *TextBox* são limpos, a *Image* é alterada e uma nova pergunta é feita. Se todas as perguntas da lista foram feitas, então deve-se recomeçar pela primeira.

A forma mais organizada e simples de se programar este algoritmo é criar e memorizar 3 listas: uma de perguntas, uma de respostas e uma de nomes das fotos. Depois "diremos" ao *App Inventor* exibir o primeiro item da lista de perguntas, mudar a *Picture* da Image para o primeiro item da lista de fotos e comparar a resposta do usuário com o primeiro item da lista de respostas. Depois faremos o mesmo com o segundo item, depois com o terceiro, até o último.

Deve ter ficado claro que precisaremos, portanto, de 4 variáveis para fazer isso, que podemos chamar de:

- Perguntas
- Respostas
- Fotos
- Índice

A variável "índice" se iniciará com o valor 1, porque queremos os itens 1 de cada lista, inicialmente, e, cada vez que o *Button* "próximo" for pressionado, este valor será adicionado em 1, para acessarmos os itens 2 de cada uma das listas; até fim da execução.

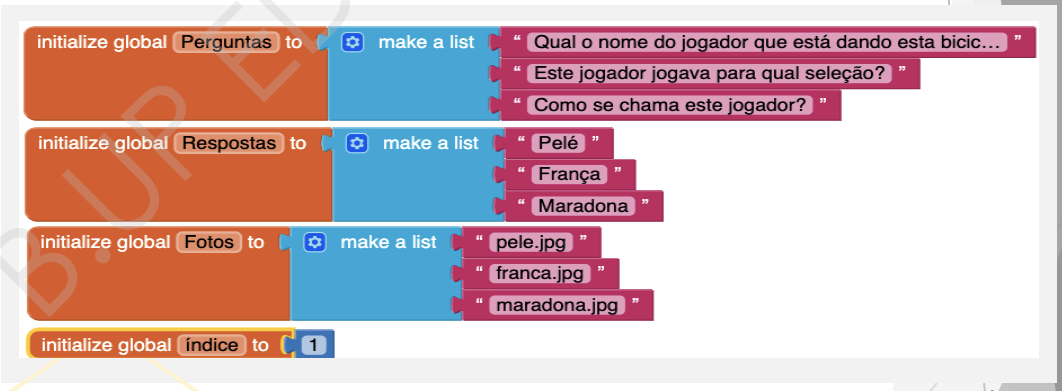


Figura 102: código

Veja o exemplo a seguir: quando *Screen* for inicializado, queremos que o texto da *Label* da pergunta seja alterada para o item "índice" da lista de perguntas e que a *Picture* da *Image* seja alterada para o item "índice" da lista de Fotos. E esta ação será repetida toda vez que o usuário quiser mudar de pergunta, logo, faremos um procedimento para programar esta ação e o chamaremos de "atualiza".

Figura 103: código

Observe: o *Button* "Próxima" deve aumentar o valor da variável "índice" em 1 e, depois, chamar o procedimento "atualiza".

Para aumentar o valor de uma variável numérica, basta definir o seu valor como sendo o seu valor atual mais 1, conforme a imagem a seguir.

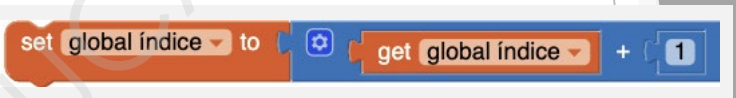


Figura 104: valor de uma variável

Exercícios

Chegou a hora de praticar! Siga os passos abaixo e salve seu trabalho.

Crie este procedimento que acabou de aprender e programe o aplicativo de modo que ele seja chamado quando iniciar o jogo e, quando o Button “Próxima" for clicado, a pergunta e a sua imagem devem ser alteradas para a seguinte.

Desafio extra

Tente agora programar o Button “OK".

UNIDADE 11

MELHORIA CONTÍNUA

PROJETO 'QUIZ ILUSTRADO II'

PROJETO 'QUIZ ILUSTRADO II'

Nesta aula daremos continuidade ao aplicativo que começamos a desenvolver na aula anterior. Vamos retomar o projeto a partir da programação do button "próxima". Ele deve aumentar o valor de índice, limpar os textos de *TextBox* e "label_certo_errado" e fazer o valor de índice voltar a 0, caso ele seja igual ao número de perguntas, para recomeçar o aplicativo num *loop* infinito. Veja o código:

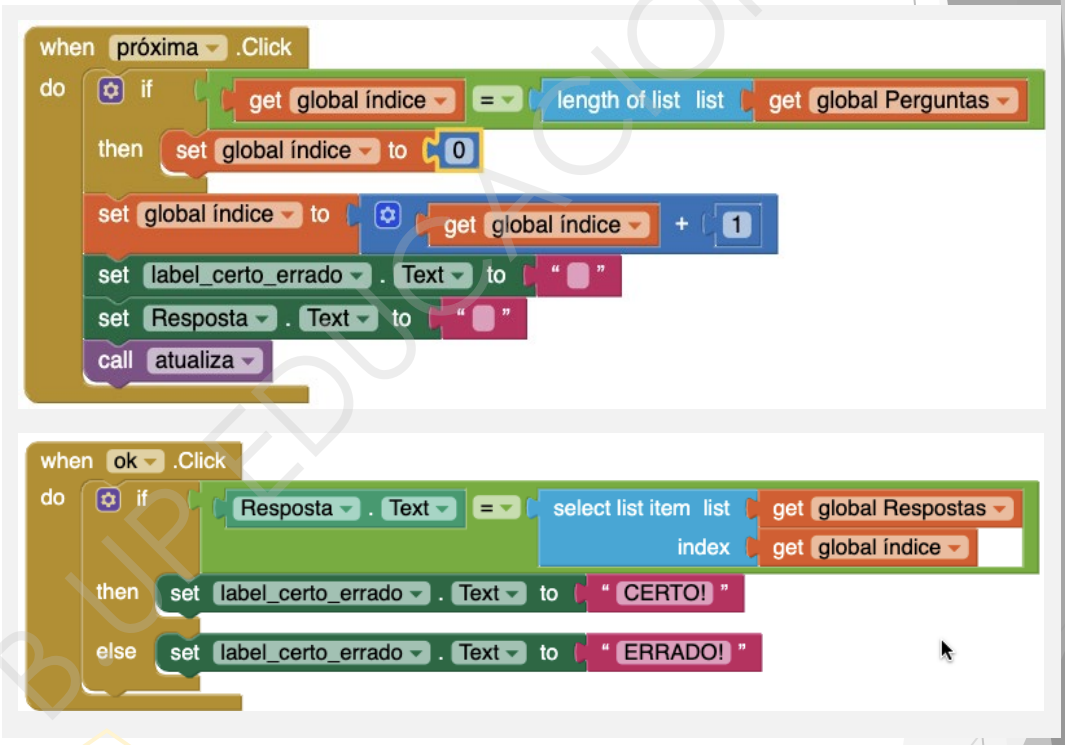


Figura 105: programação do button 'próxima'

Repare que se poderia, simplesmente, comparar o valor da variável índice com o número 3, porque temos 3 perguntas, mas se se compara com o tamanho da lista de perguntas ("*length of List* ...") não é preciso mais se preocupar em alterar este número cada vez que se colocar uma nova pergunta no aplicativo.

Só falta, agora, a programação do *Button* "OK" que avalia se usuário acertou ou errou. Basta comparar o texto de *TextBox* com o item índice de respostas.

Exercícios

Chegou a hora de praticar! Siga os passos abaixo e salve seu trabalho.

Tente o seguinte:

1. Insira efeitos de animação e rotação no componente Image;

2. Faça com que a mudança para próxima pergunta seja feita quando se clica em Image, eliminando o Button "Próxima".

3. Mude a cor do fundo da Label "label_certo_errado" para azul quando usuário acertar e vermelho, quando errar.

UNIDADE 12

LISTPICKER E ATRIBUTOS DAS LISTAS

PROJETO 'LISTA DE COMPRAS'

PROJETO 'LISTA DE COMPRAS'

Aqui vamos trabalhar algo bastante útil para a vida cotidiana: um aplicativo que cria listas de compras!

Veja a interface na imagem ao lado. A ideia básica é auxiliar o usuário enquanto ele está fazendo as suas compras no mercado. Assim, quando ele clicar no Button "Eliminar Item", irá aparecer uma lista com os itens que normalmente uma família costuma comprar no mercado e, quando usuário clicar em um destes itens, ele sairá da lista e a lista será fechada.

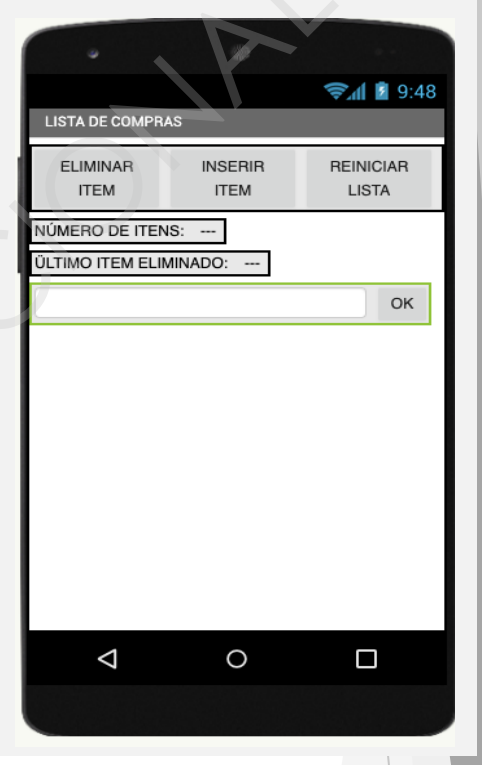


Figura 106: lista de compras

O Button "Inserir Item" serve para o usuário colocar um outro item na lista. Se caso este item já estiver na lista, então ele é simplesmente ignorado. Quando clicar no Button "Reiniciar Lista", a lista original é reestabelecida. O aplicativo ainda informa quantos itens restam na lista e qual foi o último item eliminado.

Veja os componentes utilizados neste projeto. Observe que não existem 3 Buttons, mas apenas 2. Isso porque o "Eliminar Item", na verdade, não é um Button, mas um componente chamado ListPicker, da categoria User Interface.

O último HorizontalArrangement deve ser oculto (propriedade visible, false), porque só aparecerá quando o usuário pressionar o Button "Inserir Item".

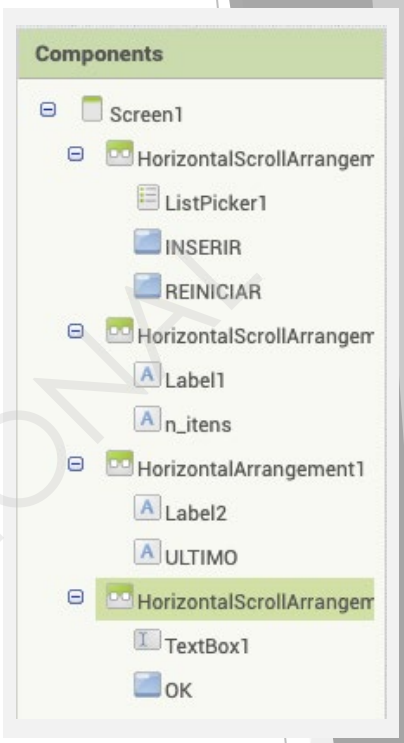


Figura 107: componentes

Componentes

1. Componente LISTPICKER (do menu 'interface do usuário')

Um botão que, ao ser clicado, mostra uma lista de textos para o usuário escolher. Os textos podem ser especificados através do *Designer* ou Editor de Blocos. Para fazer a lista no *Designer*, insira os textos na propriedade *ElementsFromString*, separando os itens por vírgula (por exemplo: choice 1, choice 2, choice 3). E, para criar a lista no Editor de Blocos, utilize a propriedade *Elements*.

Propriedades do componente LISTPICKER

- ***BackgroundColor***: Especifica a cor de fundo da *ListPicker* como um número inteiro RGB. Se uma *Image* tiver sido definida, a mudança de cor não será visível até que *Image* seja removida.
- ***Elements***: Especifica a lista de opções a serem exibidas.
- ***ElementsFromString***: Define a lista de opções separadas por vírgula.
- ***Enabled***: Especifica se o ListPicker deve estar ativo e clicável.

- ***FontBold***: Especifica se o texto de *ListPicker* deve estar em negrito. Algumas fontes não suportam negrito.
- ***FontItalic***: Especifica se o texto de *ListPicker* deve estar em itálico. Algumas fontes não suportam itálico.
- ***FontSize***: Especifica o tamanho da fonte do texto de *ListPicker*, medido em pixels independente de escala.
- ***FontTypeface***: Especifica a face da fonte do texto *ListPicker* como padrão, *serif, sans serif* ou monoespaçada.
- ***Height***: Especifica a altura da *ListPicker*, medida em pixels.
- ***HeightPercent***: Especifica a altura de *ListPicker* vertical, como uma percentagem da altura da Screen.
- ***Image***: Especifica o caminho da *Image* de *ListPicker*. Se houver uma Image e um *BackgroundColor* especificados, apenas a *Image* ficará visível.
- ***ItemBackgroundColor***: Especifica a cor de fundo dos itens da *ListPicker*.
- ***ItemTextColor***: Define a cor do texto dos itens da *ListPicker*.
- ***Selection***: Especifica o item selecionado. Quando alterada diretamente pelo programador, a propriedade *SelectionIndex* também é alterada para o primeiro item da *ListPicker* com o valor fornecido. Se o valor não estiver em *Elements*, *SelectionIndex* será definido como 0.
- ***SelectionIndex***: Método configurador da propriedade do índice de seleção.
- ***Shape***: Especifica a forma do *ListPicker*. Os valores válidos para esta propriedade são 0(padrão), 1(arredondado), 2(retângulo) e 3(oval). O *Shape* não ficará visível se uma *Image* for usada.
- ***ShowFeedback***: Especifica se um *feedback* visual deve ser mostrado quando uma *ListPicker* com uma *Image* atribuída é pressionada.
- ***ShowFilterBar***: Se *true*, o *ListPicker* mostrava uma barra de filtro de pesquisa. Esta propriedade não é, atualmente, funcional no App Inventor.
- ***Text***: Especifica o texto exibido pelo *ListPicker*.
- ***TextAlignment***: Especifica o alinhamento do texto do *ListPicker*. Os valores válidos são: 0(normal - por exemplo, justificado à esquerda se o texto for escrito da esquerda para a direita), 1(centro) ou 2(oposto - por exemplo, justificado à direita se o texto for escrito da esquerda para a direita).
- ***TextColor***: Especifica a cor do texto *ListPicker* como um inteiro RGB.

- ***Title***: Título opcional exibido no topo da lista de opções.
- ***Visible***: Especifica se *ListPicker* deve ser visível na tela. O valor é *true* se o *ListPicker* estiver sendo exibido e false se estiver oculto.
- ***Width***: Especifica a largura horizontal do *ListPicker*, medida em pixels.
- ***WidthPercent***: Especifica a largura horizontal de *ListPicker* como uma porcentagem da largura da *Screen*.

Como o próprio nome indica, iremos preencher os elementos de nossa *ListPicker* com uma lista (dos itens do mercado). Esta lista será memorizada em uma variável. Logo, a primeira coisa a ser feita é inicializar uma variável, que chamaremos de "lista_mercado", como uma lista vazia.

Figura 108: inicializando uma variável

Quando o aplicativo for aberto, 3 ações precisam ser realizadas:

1. A variável "lista_mercado" deve ser preenchida com os itens básicos do mercado;
2. A *Label* onde deve estar escrito o último item eliminado (chamada de "último") deve ser esvaziada.
3. A Label onde deve estar escrito o número de itens da lista (chamada de "n_itens") deve ser atualizada;

Para tornar nosso código mais organizado, vamos colocar todas estas ações em um procedimento chamado "INICIAR". Veja a imagem a seguir:

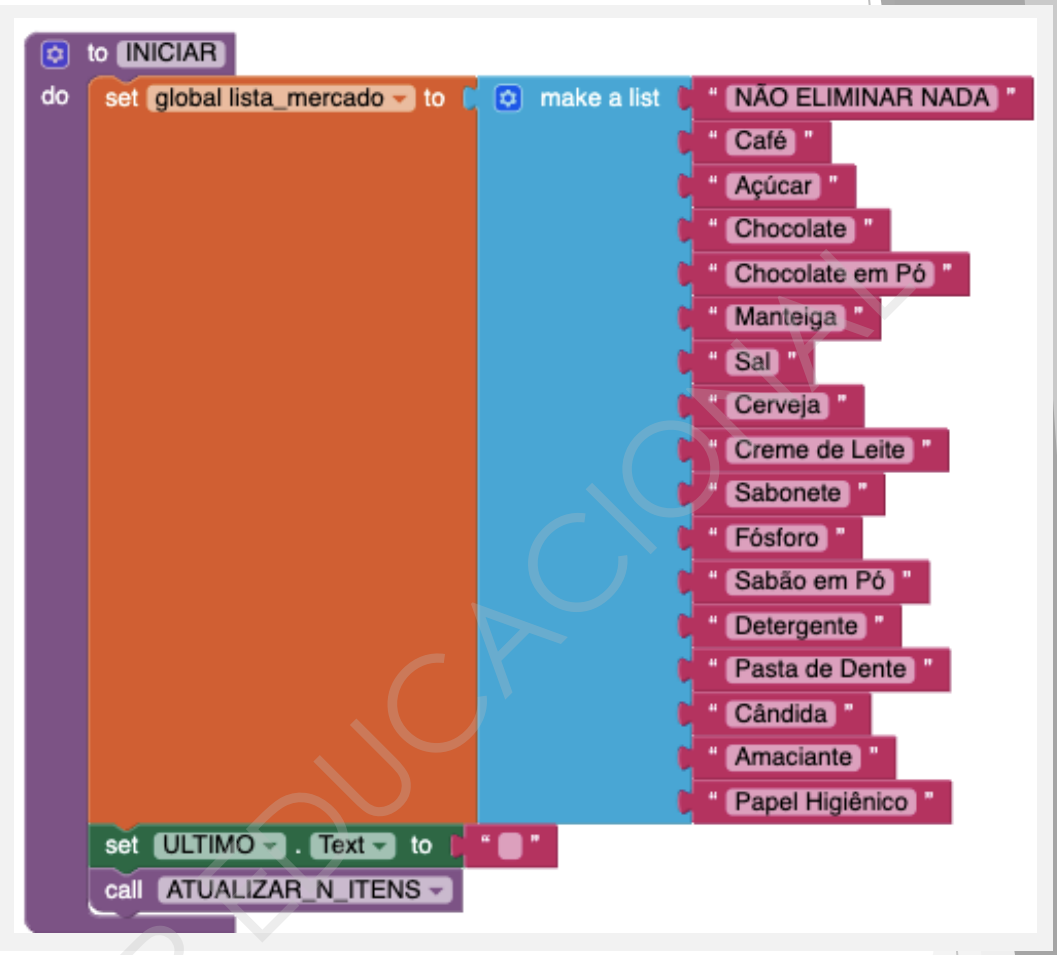


Figura 109: procedimento "iniciar"

Observe que o primeiro item da lista é a frase "NÃO ELIMINAR NADA". O motivo é o seguinte. Quando o botão do *ListPicker* é clicado, uma lista é aberta e o *App Inventor* fica esperando o usuário clicar em um dos itens; quando ele faz isso, a lista se fecha. No nosso aplicativo, o item clicado deverá ser eliminado. Mas, e se o usuário quiser fechar a lista sem eliminar nada? Por isso o primeiro item da nossa lista é esta frase. A ideia é que, se usuário clicar nela, nenhum elemento será eliminado e a lista será fechada.

Repare que a última ação do procedimento chama um outro procedimento. Isso porque esta ação será feita em outros momentos do programa (quando se eliminar ou inserir novos itens à lista) por isso é mais lógico programá-la em um procedimento próprio.

E como programar tal ação? Precisamos colocar dentro da *Label* "n_itens" o número de itens da variável "lista_mercado", ou seja, o tamanho (*length*) desta lista. Mas devemos nos lembrar de que o primeiro item de nossa lista é a frase "NÃO ELIMINAR NADA", logo, o número de itens da lista é o tamanho da variável menos 1.

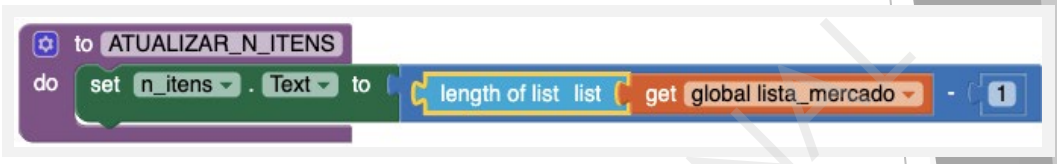


Figura 110: "atualizar n itens"

Quando Screen for inicializada, ela deve chamar o procedimento "INICIAR".

Figura 110: procedimento "iniciar"

A lista memorizada na variável "lista_mercado" deve ser atribuída à *ListPicker* toda vez que o seu botão for tocado. Ou seja, precisamos definir a propriedade "*elements*" de *ListPicker* para o conteúdo da variável "lista_mercado". Veja o código:

Figura 111: definindo a propriedade 'elements'

Exercícios

Chegou a hora de praticar! Siga os passos abaixo e salve seu trabalho.

Programe e teste o aplicativo até este ponto. Em seguida, tente programar os Buttons “REINICIAR LISTA" e “INSERIR ITEM”.

UNIDADE 13

LISTPICKER E ORGANIZAÇÃO DA ÁREA DE PROGRAMAÇÃO

PROJETO 'LISTA DE COMPRAS II'

PROJETO 'LISTA DE COMPRAS II'

Antes de continuar com a programação do nosso aplicativo, gostaria de chamar a atenção para alguns recursos que são bastante práticos, sobretudo quando se começa a produzir programas mais complexos.

O primeiro deles é a capacidade de "encolher" e "expandir" os seus blocos. Isso ocorre com o duplo clique do mouse sobre cada um dos blocos. Muito útil para tornar mais limpa a área de programação. É possível, ainda, expandir ou encolher todos os blocos de uma só vez. Para isso, clique com o botão direito do mouse sobre a área de programação e escolha a opção "*Collapse Blocks*" ou "*Expand Blocks*".

Um outro recurso útil é o "*Clean up Blocks*", também acessível com botão direito do mouse sobre a área de programação. Serve para organizar todos os blocos um sobre o outro. É possível, ainda, organizar os blocos por categoria ("*Sort Blocks by Category*").

Retomemos, agora, nosso aplicativo "Lista de Compras"!

Provavelmente você conseguiu programar facilmente o Button "REINICIAR LISTA", porque basta chamar o procedimento "iniciar", produzido na aula passada, quando o referido Button for clicado.

Figura 112: button "reiniciar lista"

Vamos, agora, programar as ações que devem ocorrer quando o usuário seleciona um item da lista. E são elas:

1. Eliminar este item da lista (desde que não seja o primeiro item: "NÃO ELIMINAR NADA");
2. Colocar na *Label* "ULTIMO" este item;
3. Atualizar a *Label* "n_itens";

Para programar a primeira ação (eliminar um item da lista) é preciso entender melhor o componente *ListPicker*. Quando o usuário clica em botão *ListPicker* e, em seguida, clica em um dos itens da lista que se abre, este item é registrado pelo *ListPicker* em um atributo chamado "*Selection*" e a posição que este item ocupa na lista é registrado no atributo "*SelectionIndex*".

Suponha, por exemplo, que em nossa *ListPicker* tenha apenas os seguintes elementos:

- Café
- Açúcar
- Chocolate

Se o usuário clicar em "açúcar", então o atributo *Selection* de *ListPicker* será "açúcar" e o atributo "*SelectionIndex*" será 2, porque o item clicado é o item 2 da lista.

Não existe nenhum bloco no *App Inventor* que remove um item da lista a partir do seu valor, mas sim a partir do seu index. É como se eu não pudesse dizer: "remova o açúcar da lista", mas só pudesse dizer: "remova o item 2 da lista."

O evento que devemos usar para executar uma ação depois que o usuário clicar na ListPicker é o "When listpicker.AfterPicking". Assim, para eliminar o item clicado da PickerList, o código é o seguinte:

Figura 113: ação 'When listpicker.AfterPicking'

E como devo fazer para que o primeiro item de nossa lista ("NÃO ELIMINAR NADA") nunca seja eliminado? Basta dizer ao *App Inventor* só executar a ação de remover item, caso o seu *SelectionIndex* seja diferente de 1, ou seja, caso ele não tenha clicado no item 1 da lista.

Para programar a segunda ação (colocar na *Label* "ULTIMO" o item selecionado), precisaremos do bloco "*select list item*", da categoria *List*, que necessita de 2 parâmetros: a lista, no nosso caso, a variável "lista_mercado" e a posição do item na lista (Index), no nosso caso, o atributo *SelectionIndex* da nossa *ListPicker*.

Finalmente, a programação da última ação desejada quando usuário seleciona o item na lista, já está pronta. Está no procedimento "ATUALIZAR_N_ITENS".

Assim, o código completo ficará da seguinte forma:

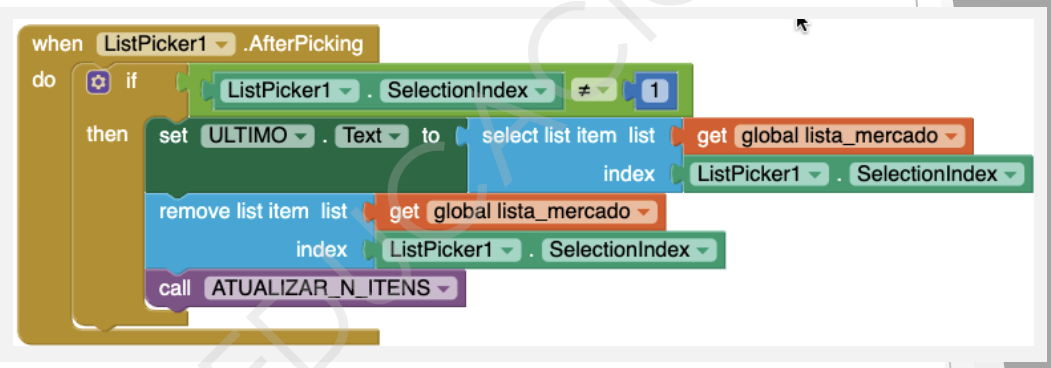


Figura 114: código completo

Só falta, agora, programar o *Button* que insere um novo item à lista. Vamos lembrar que os 2 componentes necessários para fazer esta ação, o *TextBox* e *Button* "OK" então armazenados na última *HorizontalScrollArrangement* de nosso aplicativo, que deve se iniciar invisível. Logo, a única função do *Button* "INSERIR ITEM" é tornar este componente visível. Depois que usuário escrever o novo item da lista e apertar o botão OK, este deverá tornar HorizontalScrollArrangement invisível novamente.

Figura 115: button 'inserir item'

O evento que fará com que o item escrito na *TextBox* seja inserido na lista de mercado, pertence ao *Button* "OK" ("*When OK clicked*") e o bloco que adiciona itens a uma lista é muito parecido, em sua estrutura, com o que remove itens. Ele precisa, como parâmetros, da lista cujo item será adicionado e o item que, no nosso caso, é o atributo *Text* da *TextBox*. Depois que o item for inserido, precisaremos atualizar a *label* "n_itens". Veja o código:

Figura 116: código

Existem, no entanto, duas situações nas quais não queremos que o conteúdo da *TextBox* seja inserido na lista de mercado: quando *TexBox* estiver vazia e quando já houver, na lista, o item digitado. Por isso, precisamos "dizer" ao *App Inventor* que o código acima só deve ser realizado se duas condições forem satisfeitas: textbox1.text for diferente de vazio E textbox1.text NÃO for um item da variável "lista_mercado".

Esta é uma ótima oportunidade para aprendermos a usar os operadores lógicos "E" e "NOT".

Independentemente do item digitado pelo usuário ser ou não inserido na lista, HorizontalScrollArrangement3 deve se tornar oculto novamente e a TextBox deve ser limpa. Veja o código:

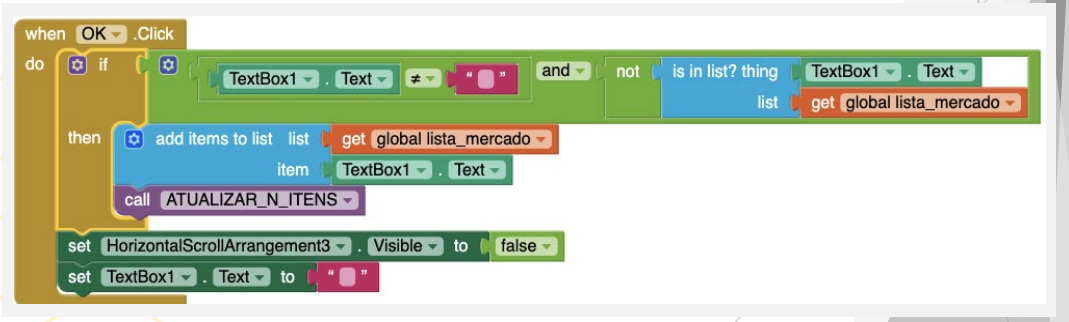


Figura 117: código completo

Exercícios

Chegou a hora de praticar! Siga os passos abaixo e salve seu trabalho.

Finalize sua lista de compras e aproveite para explorar os recursos de organização da área de programação.

Ao terminar, explore também as possibilidades de cores dos componentes utilizados para tornar o aplicativo mais agradável de se usar.

UNIDADE 14

LISTVIEW, MATRIZ E MÚLTIPLAS SCREENS

PROJETO
'QUIZ MÚLTIPLA ESCOLHA'

PROJETO 'QUIZ MÚLTIPLA ESCOLHA'

A proposta deste capítulo é a criação de aplicativo do tipo questionário de múltipla escolha. Você já possui todo o conhecimento necessário para programá-lo, usando o componente *ListPicker*, mas faremos melhor que isso. Vamos introduzir o entendimento de um novo componente, semelhante ao *ListPicker*: o *ListView*.

Componentes

1. Componente LISTVIEW (do menu 'interface do usuário')

Trata-se de um componente visível que permite colocar uma lista de elementos de texto em seu *Screen*. A lista pode ser definida usando a propriedade *ElementsFromString* ou o bloco *Elements*, no editor de blocos.

Propriedades do componente LISTVIEW

- ***BackgroundColor***: Especifica a cor de fundo da *ListPicker* como um número inteiro RGB. Se uma *Image* tiver sido definida, a mudança de cor não será visível até que *Image* seja removida.
- ***Elements***: Especifica a lista de opções a serem exibidas.
- ***ElementsFromString***: Define a lista de opções separadas por vírgula.
- ***Height***: Especifica a altura da *ListView*, medida em pixels.
- ***HeightPercent***: Especifica a altura da *ListView* de vertical, como uma percentagem da altura da Screen.
- ***Selection***: Retorna o texto existente na posição na *ListView* indicada pelo *SelectionIndex*.
- ***SelectionColor***: Retorna a cor do item quando ele é selecionado.
- ***SelectionIndex***: O índice do item atualmente selecionado, começando em 1. Se nenhum item for selecionado, o valor será 0. Se for feita uma tentativa de definir isso para um número menor que 1 ou maior que o número de itens em *ListView*, *SelectionIndex* será definido como 0 e *Selection* será definido como um texto vazio.

- ***ShowFilterBar***: Quando selecionado, torna visível a barra de filtro de pesquisa.
- ***TextColor***: A cor do texto dos itens da ListView.
- ***TextSize***: Especifica o tamanho da fonte do texto do item da *ListView*.
- ***Visible***: Quando selecionado, torna visível a *ListView*.
- ***Width***: Especifica a largura horizontal da *ListView*, medida em pixels.
- ***WidthPercent***: Especifica a largura horizontal de *ListView* como uma porcentagem da largura da *Screen*.

Vamos começar a produzir o nosso "Quiz Múltipla Escolha" com uma versão bem simples: apenas uma pergunta e 3 alternativas. Veja o layout abaixo:

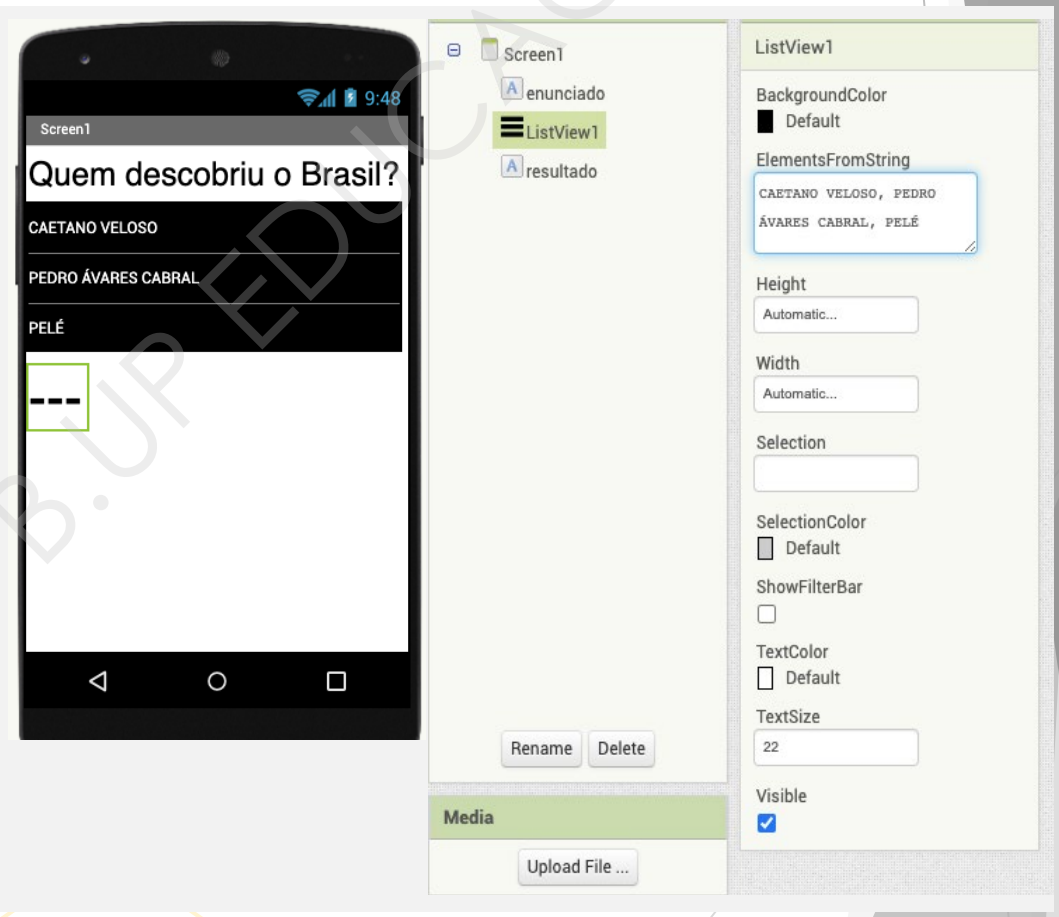


Figura 118: layout

Repare que para inserir os vários itens da *ListView* basta digitar os valores, separados por vírgulas no campo *ElementsFromString*. Vamos programar, agora, para que seja escrito na *Label* "resultado" a palavra "Certo", com um fundo azulado, se usuário clicar no segundo item da *ListView (*Pedro Álvares Cabral) e a palavra "errado", com um fundo avermelhado, se clicar em outro item qualquer.

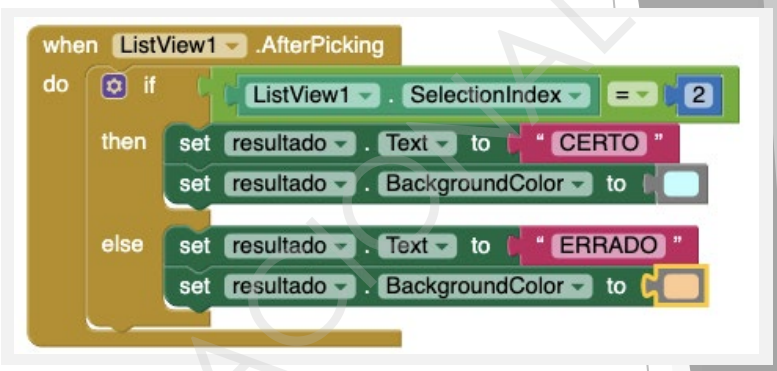


Figura 119: certo e errado

Até aqui não há muita novidade. A questão é saber qual seria a forma mais eficiente de se fazer um aplicativo com várias perguntas. Uma solução seria colocar todas as perguntas que se queira em uma lista, memorizá-la em uma variável, fazer o mesmo com as suas respectivas alternativas e inserir seus conteúdos nos componentes *Label* (para pergunta) e *ListView* (para as alternativas). Precisaremos, também, de uma variável para dizer ao aplicativo quais são as alternativas corretas para cada pergunta; e uma quarta variável onde estará registrado qual é a pergunta atual (se é a primeira, a segunda, etc).

Resumindo, precisaremos das seguintes variáveis:

1. "Perguntas", que conterá uma lista.
2. "Alternativas", que será uma lista de listas, pois, cada pergunta necessita de uma lista de alternativas (a possibilidade de se fazer listas de listas é uma novidade aqui);
3. "Gabarito", que conterá uma lista de números, cada um indicando que posição na lista o item correto ocupa;
4. "Que_pergunta", que conterá um número (inicialmente 1) que será aumentado cada vez que usuário responder a uma das perguntas.

Figura 120: variáveis

Agora faremos a programação necessária para o seguinte algoritmo:

Quando inicializar o *Screen*:

1. Deve-se escrever o item "que_pergunta" de "perguntas" na *Label* "enunciado";
2. A *ListView* deve ser preenchida pelo item "que_pergunta" de "alternativas";
3. Se o usuário clicar no item cujo *index* é o item "que_pergunta" de "gabarito", então escrever "Certo" na *Label* "resultado" e mudar a cor de fundo da mesma *Label* para azul, caso contrário, escrever "Errado" e alterar a cor de fundo da *Label* "resultado".
4. Caso a variável "que_pergunta" não seja maior que o número de perguntas, ela deve ser aumentada em 1 e o processo deve ser reiniciado.

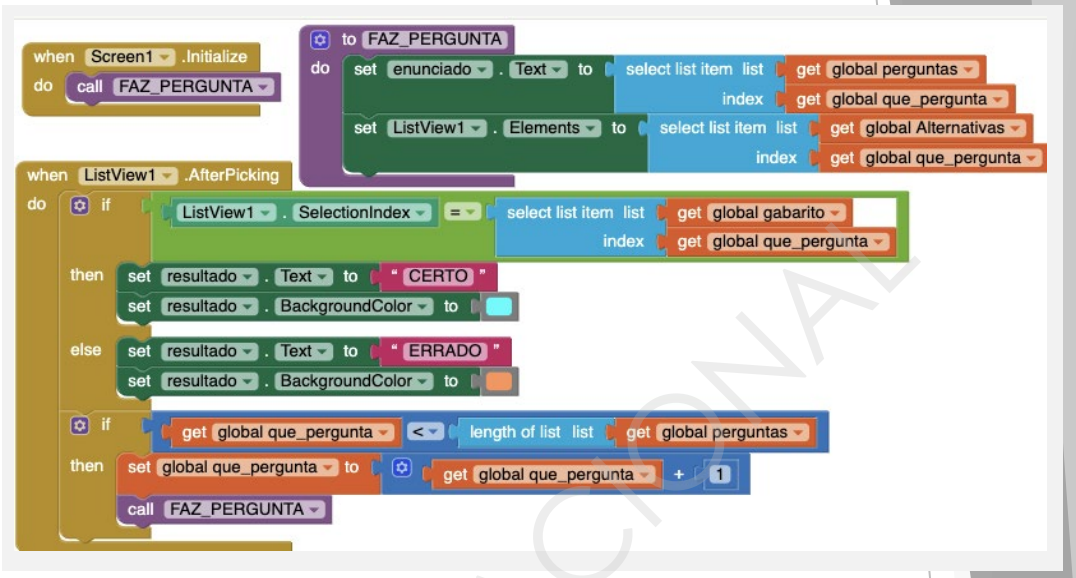


Figura 121: código possível

Este programa tem, no entanto, um problema. Como a próxima pergunta é feita logo depois que a *Label* de resultado é alterada, o usuário vê o seu resultado ao mesmo tempo em que lê a pergunta seguinte.

Uma solução seria só executar a pergunta seguinte alguns segundos depois do usuário ver se acertou ou errou a anterior, mas isso implica no uso do componente *Clock*, que será abordado em outro momento.

Por isso, optamos por uma outra solução que é a de criação de múltiplas páginas (ou *Screens*). Desta forma, quando o usuário acertar uma pergunta aparecerá uma nova *Screen*, onde estará escrito, em destaque, a palavra "Certo" e haverá um *Button* para ele continuar o jogo. Se errar, aparecerá a Screen com a palavra "Errado" e, quando terminarem as perguntas, deverá aparecer a *Screen* "Final".

Assim, nosso aplicativo, ao invés de ter apenas uma Screen, terá 5. E serão as seguintes: Capa, Screen1(Jogo), Certo, Errado e Final.

Para se criar uma nova *Screen*, basta clicar no botão "*Add Screen*" (seta vermelha), na área de menu. Esta ação solicitará que você escreva um nome para a nova *Screen*. Para navegar entre as *Screens*, utilize o botão indicado pela seta azul.

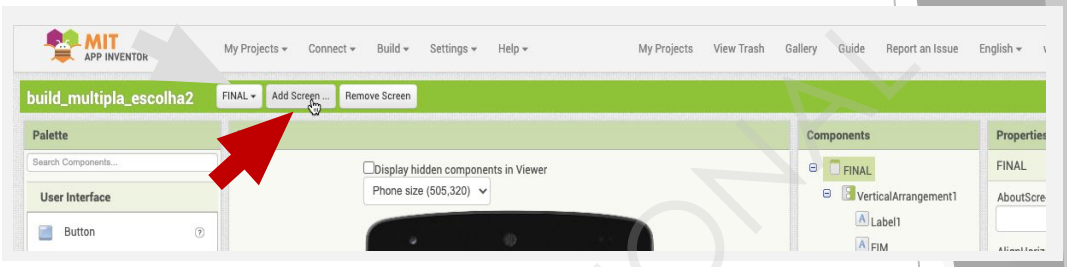


Figura 122: criando uma nova screen

Tente terminar a programação até este ponto e crie as 4 Screens restantes para o aplicativo. Seguem algumas sugestões.

Figura 123: possíveis screens

Para se programar a mudança de uma *Screen* para outra, normalmente, se utiliza o bloco "*open another screen*", da categoria *Control*. Assim, o único código da Screen capa seria o seguinte:

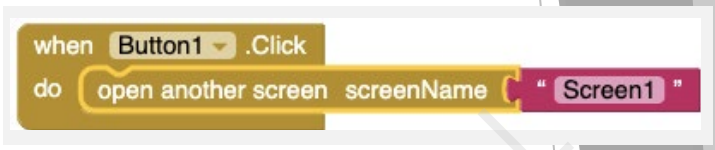


Figura 124: mudança de screen

O nome da *Screen* que se pretende abrir deve ser escrito em um bloco de texto e exatamente da forma como foi criado, diferenciando-se maiúsculas e minúsculas.

Antes de prosseguir com a programação, precisamos fazer algumas considerações a respeito das *Screens* no *App Inventor*. O ponto principal, e que pode produzir equívocos, é que as variáveis, no *App Inventor*, pertencem à *Screen* onde ela foi criada, ou seja, se você inicializar uma variável em uma *Screen*, ela não será reconhecida por uma outra *Screen*.

É possível, inclusive, criar variáveis com o mesmo nome em Screens diferentes e atribuir a cada uma delas um conteúdo diferente. E todas as variáveis de uma *Screen* são inicializadas quando se entra nesta *Screen*. Isso pode ser um problema para a programação de aplicativos como a que estamos fazendo.

Considere a situação: o usuário acertou a primeira pergunta do jogo, abriu-se a *Screen* "Certo" e voltou-se para a *Screen* "Jogo". No momento em que se volta para a *Screen* "Jogo", todas as variáveis são inicializadas, inclusive a "que_pergunta", que indica qual deve ser a próxima pergunta. Ou seja, o jogo não muda de pergunta. Felizmente, o App Inventor tem uma solução para isso.

Embora não seja possível uma *Screen* reconhecer a variável de uma outra *Screen*, é possível enviar valores de uma para outra, usando um outro bloco de transição entre *Screens* o: "*open another screen with start value*", da categoria *Control*. Este bloco exige um segundo parâmetro, além do nome da *Screen* para onde se quer ir: o parâmetro do valor que se pretende enviar.

No nosso caso específico, faremos a Capa enviar para o Jogo o valor 0. Quando Jogo for aberto, ele usará este valor para inicializar a sua variável "que_pergunta". Quando o usuário acertar a resposta, a Screen1 deverá chamar a screen "Certo" e enviar a ela o valor da variável "que_pergunta". Quando o botão da *screen* "Certo" for pressionado, ele deverá abrir a *Screen1* e enviar a ela o mesmo valor que recebeu dela. O mesmo deve ocorrer quando o usuário erra a pergunta.

Confuso, não é? Vamos passo a passo!

1. O valor 0 é enviado de "Capa" para Screen1:

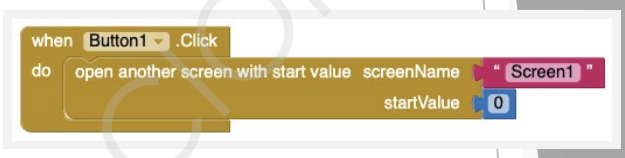


Figura 125: envio de valor 0 para screen

2. Este valor entra na variável "que_pergunta";

Figura 126: variável 'que pergunta'

3. A variável "que_pergunta" é alterada por *Screen1*, quando esta é inicializada;

Figura 127: alteração para Screen1

4. O valor "que_pergunta" é enviado de *Screen1* para "Certo" ou "Errado", conforme a resposta do usuário;

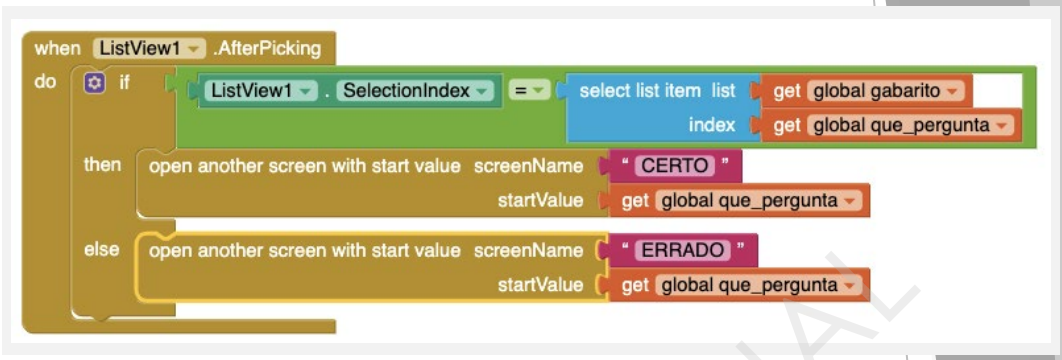


Figura 128: certo e errado

5. O valor recebido por "Certo" ou "Errado" é reenviado à Screen;

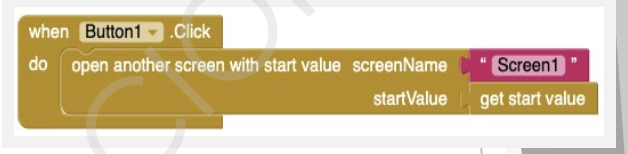


Figura 129: valor reenviado à Screen1

6. Este valor entra na variável "que_pergunta";

Figura 130: valor na variável 'que_pergunta'

A ideia básica é ficar enviando o valor da variável "que_pergunta" de uma screen a outra, já que uma *screen* não consegue acessar o variável de uma outra.

Você deve estar se perguntando por que o valor inicial, que será o primeiro valor da variável "que_pergunta", é 0 e não 1, já que deveria ser o primeiro item das listas. A resposta é simples: todas as vezes que *Screen1* é inicializada, ou seja, todas as vezes que ela é chamada, ela altera o valor desta variável antes mesmo de fazer a pergunta. Assim, se o valor inicial for 1, *Screen1* irá alterá-lo para 2 e a primeira pergunta nunca será feita.

Exercícios

Chegou a hora de praticar! Siga os passos abaixo e salve seu trabalho.

Conclua seu Questionário de Múltipla Escolha. Ao terminar, tente os seguintes desafios:

1. Insira uma imagem para cada uma das perguntas do seu aplicativo;

2. Insira recursos sonoros para indicar os erros e acertos do usuário.

UNIDADE 15

REVISÃO DE CONCEITOS E AVALIAÇÃO FINAL

'O GRANDE DESAFIO!'

O GRANDE DESAFIO!

Contexto

Você foi convidado pela empresa a participar de um *hackathon*, uma competição entre grupos para criar um aplicativo. Nessa edição, a competição tem como tema "Um app para ajudar na educação!". Crie um protótipo de app para vencer essa competição. Lembre-se de dividir bem as tarefas, pois seu app precisa ter uma boa interface de usuário, uma boa programação e ainda precisa de uma boa apresentação, pois os juízes são exigentes!

Planejamento

A primeira coisa a ser feita é um pequeno descritivo do seu app, o mais detalhado possível. Este descritivo deverá mencionar:

1. Que problema(s) o seu app deverá resolver? Em outras palavras, por que seu app será útil? Por que vale a pena gastar tempo e energia em sua construção?
2. Como funcionará seu app? Descreva-o da forma mais detalhada possível.
3. Que mídias (imagens, sons, urls, vídeos) serão necessárias para a construção de seu app?
4. Que componentes do App Inventor serão necessários para a construção de seu app?
5. Como serão divididas as tarefas da construção de seu app?
6. Cronograma de trabalho: as tarefas devem ser ordenadas e deve ser estabelecido um prazo para que cada uma delas seja concluída.

Ao terminar, apresente o seu planejamento a seu professor para que ele o valide. Provavelmente seu professor terá sugestões e correções em seu planejamento. Fique atento a elas, porque podem otimizar o seu trabalho!