

Activity 1.1.1

# Getting Started with Block-based Programming: Digital Doodle

goals



- Preview Computer Science Essentials
- Learn block-based programming
- Get started with MIT App Inventor
- Develop an app independently for creative expression

App description



**Interactive video is available in app.**

---

Create an app that allows the user to take a picture and then draw on the picture in the user interface.

## Essential Questions

- Why is it important to become a creator and not just a user?
- How does block-based programming make life easier when coding?
- Why are [independent and cooperative strategies](#) so important in computer science?

## essential Concepts

- Programming Language Abstraction
- User-centered Design
- Iterative Design and Debugging
- Event-driven Programming

## Resources

Independent and Cooperative Strategies

Computer Science Practices

PLTW Computer Science Notebook

App Inventor Debugging Guide

1.1.1 Challenge: Additional App Features

Digital Doodle app icon



## Getting Started with MIT App Inventor



Design Overview: Digital Doodle

**App Overview** You are going to create an app that allows a user to take a picture and then draw on that picture. For replay value, the device screen will clear whenever a user shakes the device. Each of these items is called an app feature, because it describes a specific thing that the app may do. An abbreviated tutorial for this app and many others can be found at the [MIT App Inventor website](#).

**User Story** Each feature in an app can be described as a user story. User stories define and prioritize the work you do. The user stories for the Digital Doodle app include three user interactions:

- An app event that allows a user to take a picture when they touch a button.
- Another event that allows a user to touch the screen (swipe gesture as input) to draw on the picture.
- A final app event that allows the shaking of the device to clear all the outputs (the drawing the user made on the picture).

**Initial Backlog Breakdown** The user needs to be able to:

- Push a button to take a picture
- Draw on the picture
- Clear the picture
- Change the color they draw with (if time permits)
- Change the line width (if time permits)

Design Terminology

---

### User Story

Each of the app features is a user story. User stories are the individual items that make up the whole solution or app. Developers make a plan for development that prioritizes and individually addresses each user story's need. With each feature you add, it is important to test and get feedback from other users on how your idea or solution is working. Sometimes new user stories will emerge as people think of new things they want the app to do. All these features need to be visible to the user in the user interface.

### User Centered

When you are developing an app, or any software solution, it is important to think about the people who will be using that software. If the design and development of a software solution is not user centered, then the app may not be used by the intended audience.

### User Interface

A user interface is what the user interacts with on the device. It can include the touch screen, buttons, and even an accelerometer that senses and sends the orientation of the device. The user interface sends messages to the operating system to indicate what to do next based on what it senses from the user. As you design apps, you will create many types of user interfaces to meet different user story needs.



**PLTW COMPUTER SCIENCE NOTEBOOK -Design Terminology** As you progress, take note of the vocabulary words using a [TEMP Chart](#) (Term, Example, Meaning, Picture). As you come across additional key terms, add them to your TEMP chart. Review an example TEMP chart on the [PLTW Computer Science Notebook](#) page.

Add the words in the previous Design Terminology section into a TEMP chart.

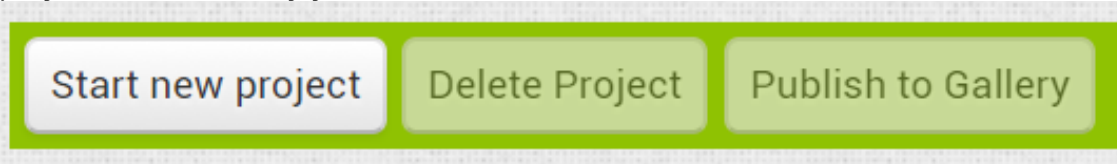
## Setting Up the Account and Project

---

Access MIT App Inventor in a web browser on your computer.

- a. Navigate to <https://accounts.google.com> and log in to your Google account.
  
- b. Navigate to MIT App Inventor at <http://appinventor.mit.edu/explore/>
  
- c. Click the button that says **Create apps!** and allow login with your Google username and password.
  
- d. Bookmark the MIT App Inventor website on your computer as directed by your teacher. Because you will access this site every class for the next few weeks, this will save you some time at the start of class.
  
- e. On your Android™ device, check to see whether the AI2 app is installed. If not, navigate to the Google Play Store at <http://appinv.us/companion> and download the AI2 Companion App.

After App Inventor is open in the web browser, select **Start new project**. Name the project as directed by your teacher.



My Projects Menu

## Designer View and Blocks View

There are two types of views in App Inventor:

*Designer* view, where you can create the user interface and add common features you want in the app. This is where you let the program know what components you will later want to code.

- *Blocks* view, where you can program the features you added in the *Designer* view. To open the *Blocks* view, click **Blocks** in the upper-right corner of the window.

While you develop apps, you will switch between the views. To know which interface to work in, think to yourself:



**Interactive flashcard is available in app.**



**Interactive flashcard is available in app.**



**PLTW Computer Science Notebook -Views** Add the two types of views to your TEMP chart.

## Components in Designer View

---

Within the *Designer Palette*, you will find *drawers* with many different *components* in them.



### Designer View

A **component** is a tidy package of functionality for an input or output. *Components* are in the *Palette* on the left side of the *Designer* view. The *Designer Palette* has different *drawers* that you can open to see *components* in each category, such as the *User Interface* drawer or *Sensors* drawer.

Toward the right side of the window is a *Components list* of all the components you have added to your app from the *DesignerPalette*. In the *Components* list, you can select, rename, or delete the components you want. When you select a component from the list, *Properties* on the far right changes to provide options for the component you selected.



Remember from the user story that this app will need the following components:

- Button (to activate the camera)
- Canvas (to draw on)
- Camera (to take the picture)



**Error Alert:** You need to have these three design components in the *Designer* view, so that you can switch to the *Blocks* view to program the components. Without the design components in place, you will not see the blocks to make the program.

Drag the three components onto the user interface of your app from the following drawers:

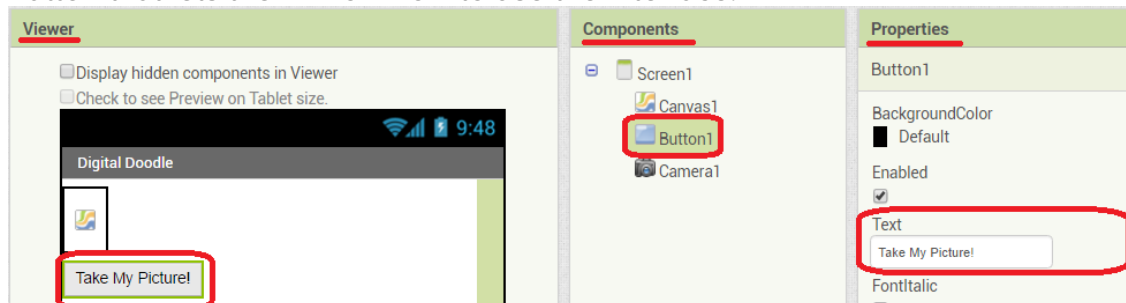
- a. *User Interface* drawer: Drag out the *Button*, which responds with programmed actions when a user clicks it.
- b. *Drawing and Animation* drawer: Drag out the *Canvas*, which provides a touch-sensitive surface where a user can draw and interact.

c. *Media* drawer: Drag out the *Camera*, which accesses the device’s camera. The camera is a non-visible component, so it will appear at the bottom of the window when you drag it to the *Designer* view.

The default *Text* setting for the *Button* component is “Text for Button”, which does not help the user know what the button does.

In the *Designer* view, select the **Button** in the *Viewer* or *Components* list.

Change the *ButtonText* property to **Take a Picture!** Now the user will see text on the *Button* that lets them know how to use the interface.



Designer View Updating Properties



**Interactive slideshow is available in app.**

---

## Event Handlers

In almost all programs you create, you will have inputs or events that cause the program to take action. These actions usually produce outputs that the user can experience. Events include actions such as clicking a button, touching a screen, or tilting a device that has an accelerometer in it. The program might produce an output for the user, such as sound, graphics, or motor movement. Sometimes the program does not give a noticeable output to the user, but changes something in the program.

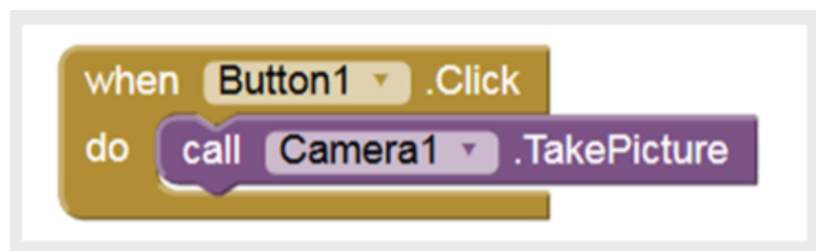
Event handlers look for inputs or events to know when to perform a specific action and provide specific outputs. Some event handlers are control blocks that are specific to a component in App Inventor.

Event handlers abstract the details of the structures that enable or disable the control over the components you drag and drop inside them. Each block in App Inventor is doing a lot of work for you behind the scenes. Abstraction keeps you from seeing, and dealing with, the details you don't care about (for right now).

In this example, the control block, or *event handler*, is waiting for an event (*Button* click) and a procedure that produces an output (takes a picture).

### Event Handler

when: *Button1* is clicked (input)do: call *Camera1* and *TakePicture* (output)



### Event Handler

Snippet of code showing the event handler for the first feature in the Digital Doodle app. It will use the input of pressing a button to take a picture.

As a developer, you would waste time writing the same code to take a picture every time you use that feature in an app. So that code is bundled in a procedure block for repeated use, without having to recode it each time.

App Inventor abstracts away those details, so you can focus on the app as a whole and use a camera feature, instead of all the steps needed to make the app take a picture.

Abstraction hides the complexity of a task by concealing the details, making it easier for computer scientists to focus on the relevant steps of creatively developing code.

## Blocks View

---

While developing in the *Blocks* view, pay close attention to the event that you want to develop. To find an event handler, look at the event inside the event handler, then find that event component in *Blocks* on the left side of the *Blocks* view.

To switch to the *Blocks* view, click **Blocks** in the upper-right corner.

In the *Blocks* panel on the left, find the blocks below and drag them out into the *Viewer* part of your screen:

*Camera Drawer*

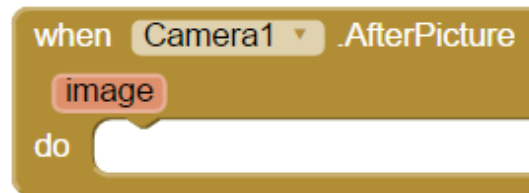


A block that calls a procedure that lets the user take a picture.



A block that sets the background image of the canvas whenever the event handler the block is placed in is activated.

### ***Canvas Drawer***



A block that handles the events after the user takes the picture.

### ***Camera Drawer***



An event handler that executes the blocks within it whenever *Button1* is clicked.

### ***Button Drawer***

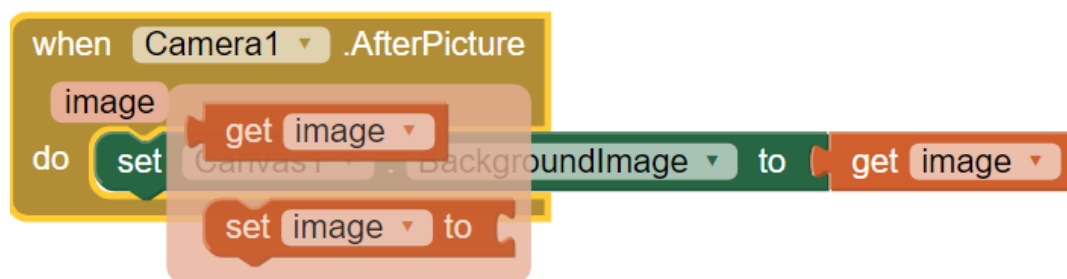
If you cannot find a block described in the procedures:

- Verify that the component is in the *Designer Components* panel.

- If the component is not in the *Designer* view, add it in the *Designer* view then return to the *Blocks* view.
- Find the same component in the *Blocks* view to be able to grab blocks associated with that specific component.

Drag out the image block from the *Camera1.AfterPicture*.

- a. Move your mouse pointer over the text “image” without clicking.
- b. Click the *get image* block that pops up and drag it out. The image block may only be used inside the event handler you pulled it from.



**Interactive show hide is available in app.**

---

Setting Up Event Handlers

---

An event handler is a chunk of code that executes (is put into action) when a particular event occurs. To create an interface with an event handler in MIT App Inventor, drag one or more blocks into the event handlers.

Look at the “*When..., do...*” *Button.click* event handler block.

Each time the event occurs (selecting the button on the screen), the code inside the *do* part of the event handler will be performed.

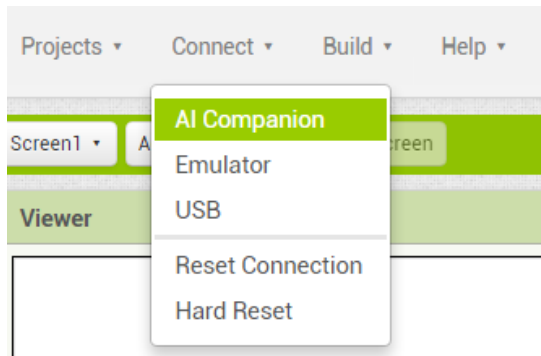
The blocks work like a puzzle. Either the blocks fit together, or you need to revisit how you are using the blocks to code.

Construct two event handlers from the blocks you pulled onto the screen by thinking through two of the features you want:

- a. What event(button) triggers an eventhandler (button click) to make what action happen (call the camera to take a picture)?
  
- b. After the picture event, the camera should set the *Canvas* to show the background as that image from the camera.

# Debugging

To test your program, connect to the device with the following steps:



Connect Menu

- a. In the MIT App Inventor browser window, select **Connect > AI Companion**.
- b. On the Android device, launch the **AI2 Companion** app. The browser displays a six-character code.
- c. In the **AI2 Companion** app, you may connect in two different ways:
  - i. Companion app, enter the six-character code and select **Connect with code**.
  - ii. Alternatively, you can scan the QR code in the Android device's AI Companion app.



Your program is working properly, if you can:


- Touch the button as input
- Receive the output of an image on the screen after you take the picture

**Important:** Check the boxes as you verify parts of the app are working. As you test, this will help you know what works and what does not.

If your program is not working properly, start debugging, which means looking at your code piece by piece to determine why the program is doing something different from what you intended it to do. Refer to the App Inventor Debugging Guide for reminders and steps to debug.

## App Inventor Debugging Guide

Be patient with yourself, and seek help from those around you if your app is not performing as expected.

 **PLTW Computer Science Notebook - Debugging** Title a page “Debugging”. Use this page to record information and tips about debugging your code.

## Iteration and Version Control

After testing and debugging, you have a functioning app!

You probably want to keep it that way as you explore adding more features. Saving your project as a new version (with a new name) will help minimize how much you have to debug or fix moving forward. Naming is important, because it will help you review previous versions of your code as you develop toward the final app.

To save your program with a new name, click to open the **Projects** menu and select **Save project as...**

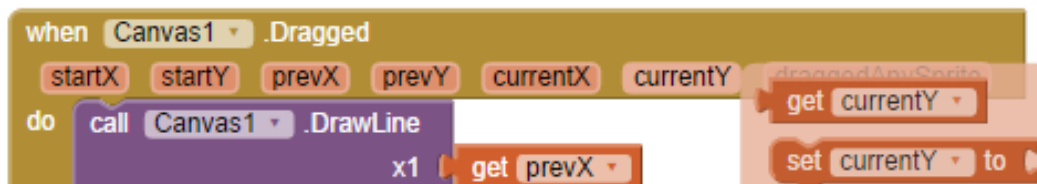
- a. As you create these new versions—also known as program iterations, it is important to name them in a way that will make it easy to find specific features or code.
  
- b. Discuss naming schemes with your teacher.

## Procedures and Arguments

You will set up a `callCanvas.Drawline` procedure under the `Canvas.Dragged` event handler.

In the *Canvas* drawer, drag out the *callCanvas.Drawline* and the *Canvas.Dragged* event handler into the *Blocks Viewer*.

The *whenCanvas1.Dragged* event handler block is designed to automatically get (X,Y) location information for you to use in your program (both the previous location and current location when a user drags their finger across the screen). Hover over the orange arguments in the event handler (as pictured below) to access the *get variable* blocks that contain the (X,Y) values. The program can pass the information about the (X,Y) locations to other parts of the program as arguments/variables by plugging a *get* block into the *callCanvas1.Drawline* procedure.



### Event specific arguments

For the app to draw a line, the procedure needs to get the coordinates of the screen. The app needs to know where the user first touched and where they touch next.



**Interactive slideshow is available in app.**

---

These types of blocks will be discussed more later.

Set up the procedure while considering the previous pictures. For help with the setup, you can read the comments in the figure above, hover your pointer over the event handler in your program to see an explanation, or read more below.



## Interactive show hide is available in app.

---

Test and debug your app.

- a. If you change your app and it no longer works, you may always go back to the previously saved version.
  
- b. Do an iteration save of your project, if the app allows you to:

- Take a picture
- Draw on the picture

### Clear Screen

---

As you test the app by drawing on it, you may want to clear the screen. It is possible to clear the canvas when you shake the device using the accelerometer. The accelerometer is a built-in sensor in some devices that allows the device to sense if and how much the device is tilted in any direction.

To use the accelerometer in the app:

- a. In the *Designer* view, go to the *Palette Sensors* drawer and drag in the *AccelerometerSensor* component. The *AccelerometerSensor* is a non-visible component, so it will show under the app screen in the *Viewer*, but the user cannot see it.
  
- b. In the *Blocks* view, select the **AccelerometerSensor** drawer and drag out the *AccelerometerSensor.Shaking* event handler.
  
- c. In the *Blocks* view, select the **Canvas1** drawer. Find and drag out the call *Canvas1.Clear* procedure block and place it inside the *AccelerometerSensor.Shaking* event handler.
  
- d. Test, debug, and save when the app lets you:
  - Take a picture
  - Draw on the screen
  - Shake the device to clear all the drawings

Add Color

---

The app needs a little color. You can add buttons that will change the pen color. Now is your chance to add the colors you think a user may want to draw with.

To add color buttons:

- a. In the *Designer* view, add another button.
- b. Select the new button in the *Components* list. In the *Designer Properties* of the button, change the *TextColor* to the color you want it to be, for example *red*.
- c. In the *Blocks* view, click the button that you want to be red. Find the *WhenButton.click* event handler and drag it out into the screen.
- d. Click the **Canvas** drawer to find the *set canvas.PaintColor* block and add it to the event handler.
- e. Click the **Colors** drawer and pull out the color and plug it into the end of the set block.
- f. Test your app:

Take a picture

Draw on the screen

- Select different colors to draw on the screen
- Draw on the screen
- Shake the device to clear all the drawings

## App Completion

---

Review the iterations you have completed with your code.

---



**Interactive sideline is available in app.**

---

When your app is working properly, complete the following as your teacher directs.

a. Share your work.

- Show your MIT App Inventor screen.
- Demonstrate the app on a device using AI Companion.

b. Back up your work.

- To download the MIT App Inventor program you created, select **Projects > Export selected project (.AIA) to my computer**.
- To download the Android app, select **Build > App (save .APK to my computer)**.

c. Share a quick reference image.

- In the *Blocks* view, right-click on the blank area of the screen.
- In the popup list, select **Download Blocks as Images**.
- Share the image with your teacher.

## Challenge

Challenge yourself by choosing some [additional app features](#).

## Conclusion

Why is it important to design incrementally? Consider: During what iteration did you have a working app? What did you stand to lose and gain with each iteration?

Why do you think computer science professionals included things like communication, collaborations, and fostering an inclusive computing environment in the [computer science practices](#)?



How did you interpret and respond to the [Essential Questions](#)? Capture your thoughts for future conversations.

Proceed to next activity

# App Inventor Debugging Guide

## Resources

### Steps for Debugging Code



- Explain to an elbow partner where you are in the activity, what you were trying to do, and what you have already done. Sometimes talking through these steps helps you discover the problem.
- Read any errors that pop up on the screen. These will lend insight into why the software cannot complete the program.
- Remove or right-click to disable some blocks to get a basic working code, and then add blocks back in one at a time to see what is causing the issue.
- Check the names of components in the *Designer* view to make sure the blocks you think you are programming are the ones you are actually programming.

### Steps for Debugging Code



Check for hardware or emulator connection issues. You might need to close the emulator or the MIT AI Companion, reset the connection, and reconnect.



Check for compilation errors (warnings and error messages in the *Blocks* view) and fix them.



To make sure you understand the intended outcomes of each event handler, review your comments and algorithms.



Make sure your code is easy to read. Collapse code that is not related to the bug.



Use debugging strategies to isolate the bug.

Do It command

Disable command

Code trace or variable trace



Test the app.

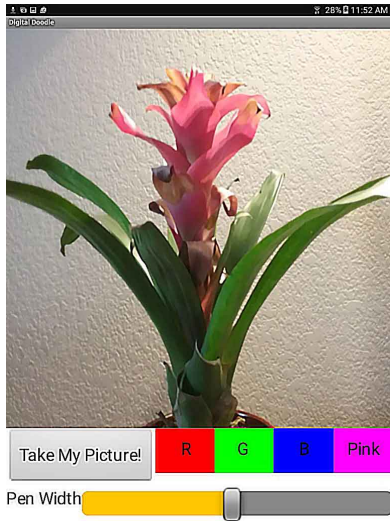


Fix the bug.



Test the app again.

# Challenge: Additional App Features



## Sample User Interface

Your teacher will direct you about which additional features to add. Review the goals, designer needs, and what to do to add these features.

To make sure you do not lose progress on a working app as you add new features, use iterative saves and the naming convention outlined by your teacher.

## Challenge A - Pen Size Feature

### Goal

Add a slider to adjust the width of the line that draws as you drag your finger on the touch screen.

### Design Needs

- *Horizontal arrangement*, which lets you adjust where items are placed in comparison to each other on the screen.

**Important:** Drag in the *horizontal arrangement* first so that you can drag the color buttons directly into the horizontal arrangement, side by side.

- *Label* component, to identify the pen slider to a user.
- *Slider* component, to change the value as the user slides a control back and forth.

Do

- The following image shows a *slider* component beside a *label* component inside a *Horizontal Arrangement* component, as you might see in the *Designer* view screen.



**Important:** While you have the horizontal arrangement selected, look under the *Properties* on the right of the screen. Where it says “Width”, change that option to **Fill parent** to make it go all the way across the screen.

Repeat with the slider to make it a more user friendly [input](#)control.

- Take special note of the argument that the *slider.positionChanged* event handler offers.
- What component are you setting up to accept that argument?
- Once it is working, save your project, adding an “A” to the end of the name to show you completed Challenge A.

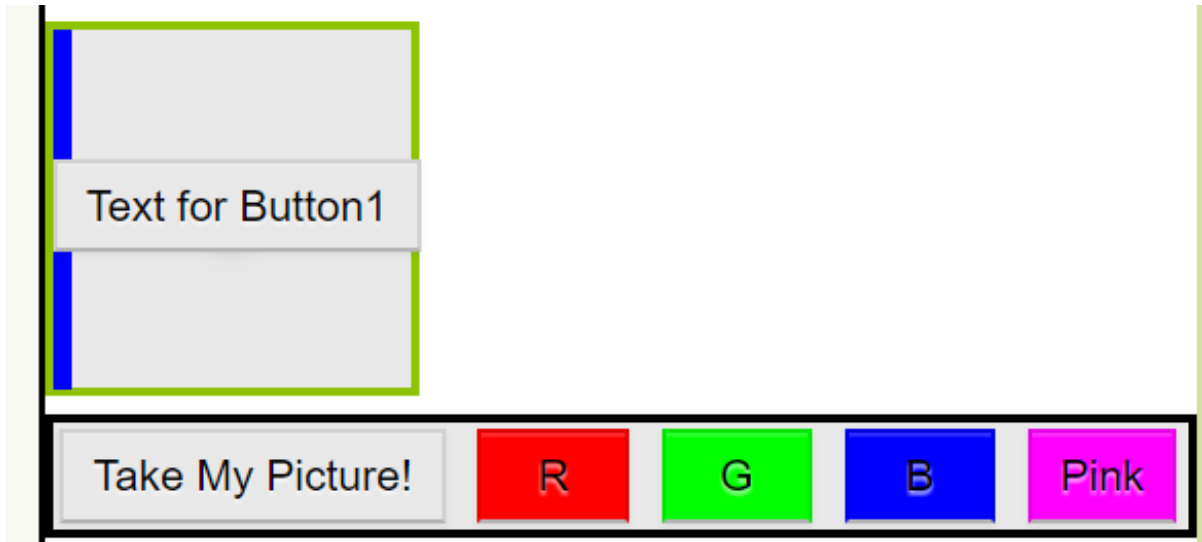
## Challenge B - Color Change Feature

Goal

Add buttons to change the color of the pen.

## Design Needs

- Horizontal arrangement
- Three buttons inside a horizontal arrangement



### Horizontal Arrangements

The top block in the figure shows what a horizontal arrangement looks like when you add it to the *Designer* view screen. Drag items into the block and watch for the solid blue line (as shown above) to indicate where the component will be placed inside the horizontal bar. An example of a completed horizontal arrangement is shown at the bottom of the image.

Do

- Set the *Shape* and *BackgroundColor* properties of each button.

**Important:** In the *Components* list of the *Designer* view, you should rename each button to a name like *ButtonBlue*. That helps programmers keep track of the components.

- Once it is working, save your project, adding a “B” to the end of the name to show you completed this challenge.

## Challenge C - Add a Fourth Button

### Goal

Add another button with a custom color.

### Design Needs

Another button - be sure to adjust the size, color, and name of the button.

### Do

- Add a fourth button, and instead of using a color block, use the blocks in the image.



- To see what colors you can make, change the values of the red (first number), green (second number), and blue (third number) .
- Once it is working, save your project, adding a “C” to the end of the name to show you completed this challenge.

## Challenge Iterations

---



**Interactive slideline is available in app.**

---

## Continue Exploring

Explore the other components and choose one feature to try out in this app. For a list of components and information about them, check out the App Inventor [Component Reference](#).