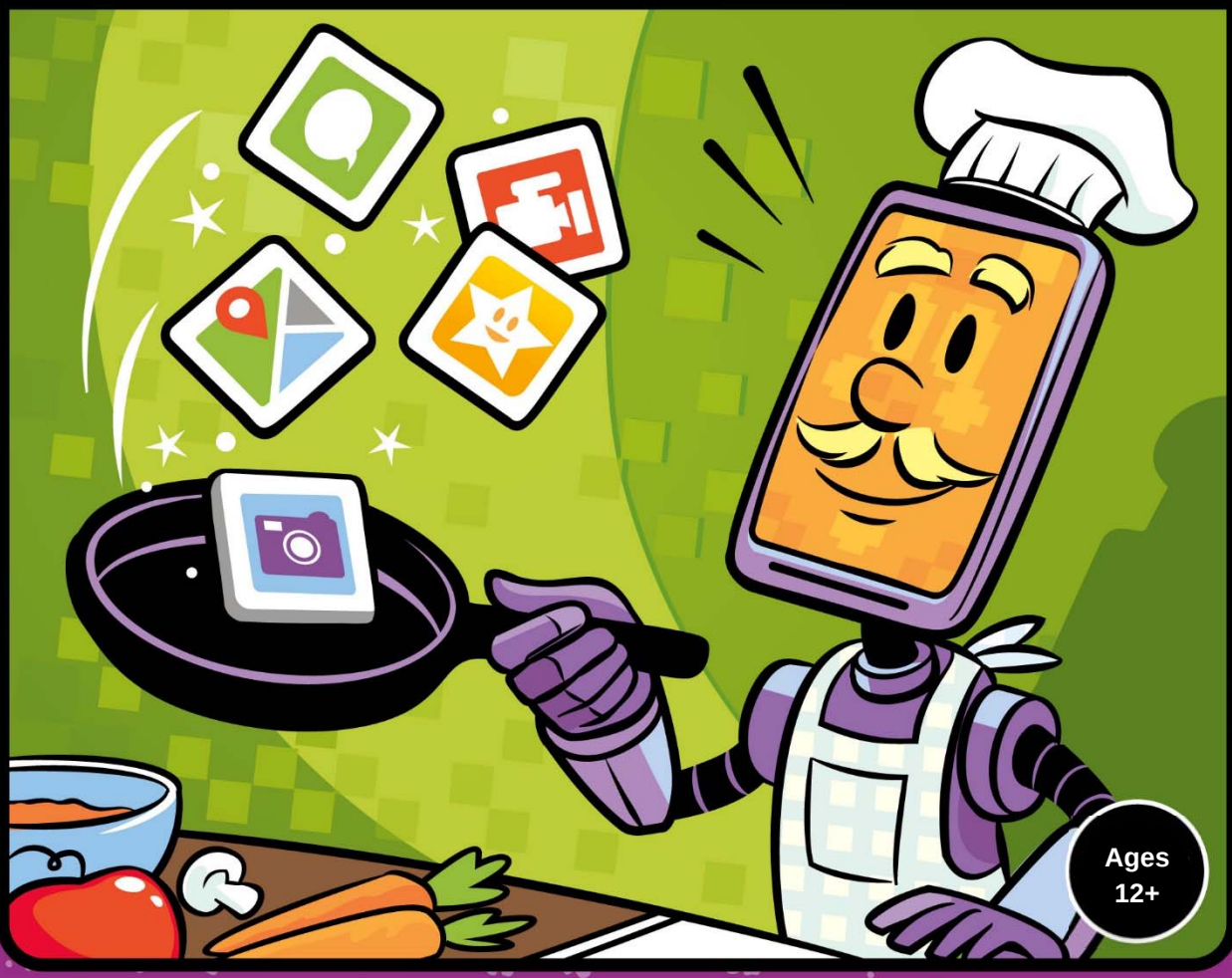
LEARN TO PROGRAM WITH APP INVENTOR

A VISUAL INTRODUCTION TO BUILDING APPS

"ON YOUR OWN" SOLUTIONS



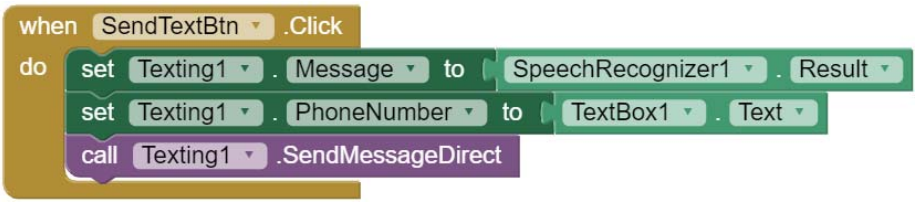
Ages 12+

CHAPTER 1 - ON YOUR OWN EXERCISES - POSSIBLE SOLUTIONS

1 – Change the *Hi, World!* app so that it prompts the user to enter a telephone number rather than selecting one from the contact list. Which component(s) would you drag from the User Interface drawer to the Viewer to enable the user to enter the telephone number manually? Now that you'll be replacing the `PhoneNumberPicker` component, how will you include instructions to the user about where to enter a phone number? What blocks will you use to provide the phone number required for `Texting1`?

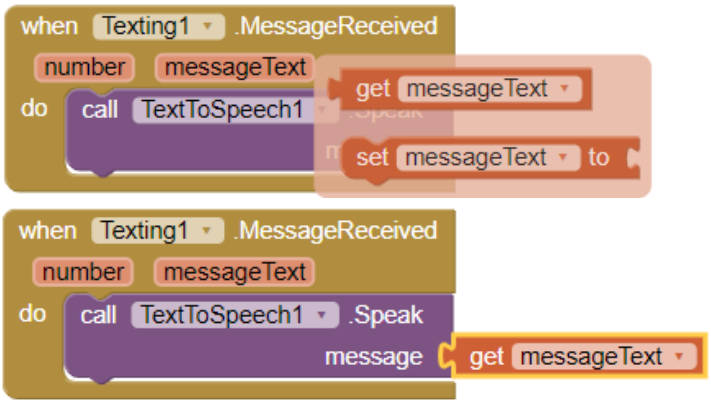
a. In the Designer, replace the `PhoneNumberPicker` component with a **`Label`** and **`TextBox`** from the User Interface drawer.

b. In the Blocks Editor, set the phone number for `Texting1` to the text entered by the user into `TextBox1` (use the **`TextBox1.Text`** block).



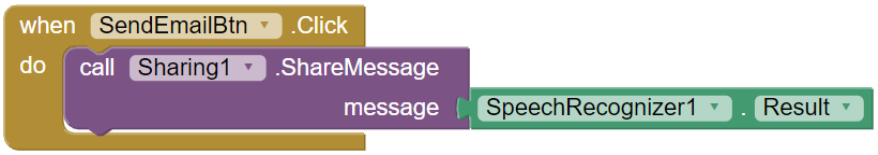
2 – Extend the app for Android devices so that, after it texts the spoken message, it waits for a response text message from the recipient and then reads that message to the user aloud.

a. In the Designer, add a non-visible **`TextToSpeech`** component from the Media drawer.
b. In the Blocks Editor, click **`Texting1`** and drag in a **`whenTexting1.MessageReceived`** block. Click **`TextToSpeech1`** and drag the **`call.TextToSpeech1.Speak`** block into the **`whenTexting1.MessageReceived`** block. Mouse over `messageText` on the **`whenTexting1.MessageReceived`** block and snap its **`get messageText`** block into the open **`message`** socket.



3 – Change the app so that it emails the message instead of texting it. What components and blocks would you use to send an email message?

a. In the Designer, remove the `PhoneNumberPicker` component, rename `SendTextBtn` to **`SendEmailBtn`**, and replace the `Texting` component with a **`Sharing`** component from the Social drawer.
b. In the Blocks Editor, click **`Sharing1`** and drag a **`callSharing1.ShareMessage`** block into the **`whenSendEmailBtn.Click`** block. Click **`SpeechRecognizer1`** and snap its **`SpeechRecognizer1.Result`** block into the **`callSharing1.ShareMessage`** block's open **`message`** socket. This will let the user share the spoken message (the `SpeechRecognizer1.Result`) with the device's email application.

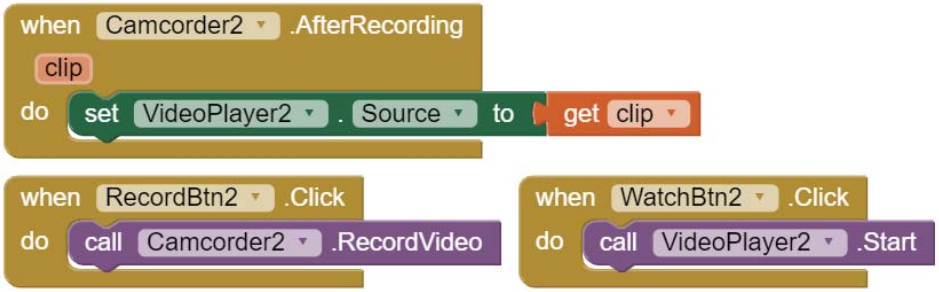


CHAPTER 2 - ON YOUR OWN EXERCISES – POSSIBLE SOLUTIONS

1 – Extend the *Practice Makes Perfect* app so that the user can record, watch, and compare two practice video clips side by side. Which Layout and other component(s) would you need to drag to the Viewer to enable this? How will your blocks change?

a. In the Designer, for the second practice video, add another record **Button** and watch **Button** from the User Interface drawer and another **Camcorder** and **VideoPlayer** from the Media drawer all into VerticalScrollArrangement1. So that you can place both VideoPlayers side by side, drag a **HorizontalArrangement** from the Layout drawer to the bottom of VerticalScrollArrangement1, and place **VideoPlayer1** and **VideoPlayer2** inside HorizontalArrangement1.

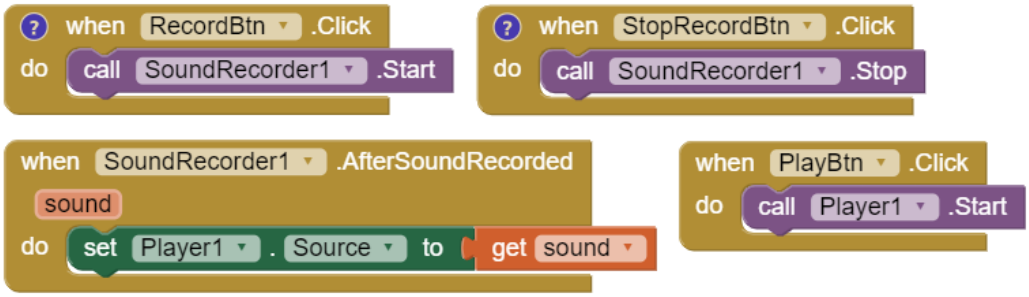
b. In the Blocks Editor, program the **Camcorder2 AfterRecording** event handler to set the source for VideoPlayer2 to the clip recorded by Camcorder2. Also, program the second record **Button Click** event handler to open Camcorder2 and the second watch **Button Click** event handler to start VideoPlayer2.



2 – Change the app so that it records and plays sound clips instead of videos. What components and blocks would you use to record and play sound?

a. In the Designer, add a **Button** from the User Interface drawer into VerticalScrollArrangement1 under RecordBtn and rename it **StopRecordBtn**. Rename WatchBtn to **PlayBtn**. Replace the Camcorder and VideoPlayer components with a **SoundRecorder** and **Player** from the Media drawer.

b. In the Blocks Editor, program **RecordBtn** to start the SoundRecorder when clicked and **StopRecordBtn** to stop the SoundRecorder when clicked. Program the **SoundRecorder1 AfterSoundRecorded** event handler to set the source for Player1 to the recorded sound. Program **PlayBtn** to start Player1 when clicked.

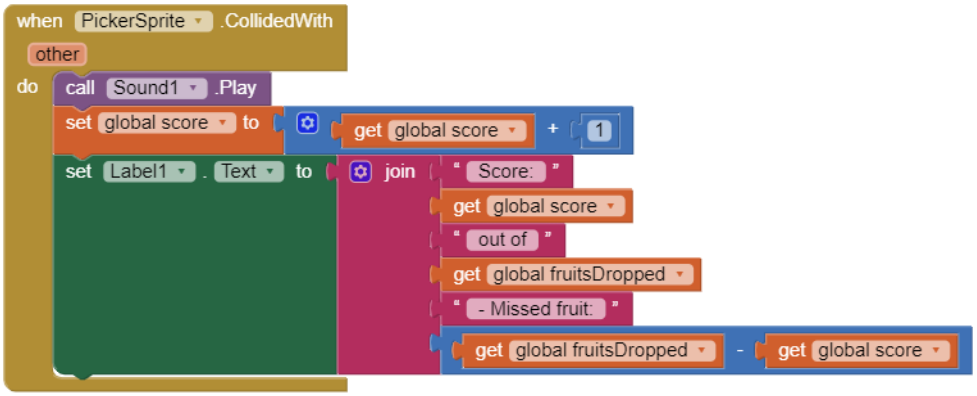


CHAPTER 3 - ON YOUR OWN EXERCISES – POSSIBLE SOLUTIONS

1 – Change the *Fruit Loot* app so that it calculates and keeps track of how many pieces of fruit the picker fails to catch during a game. How can you calculate, store, and display this information using the existing event handlers and adding the smallest number of additional blocks?

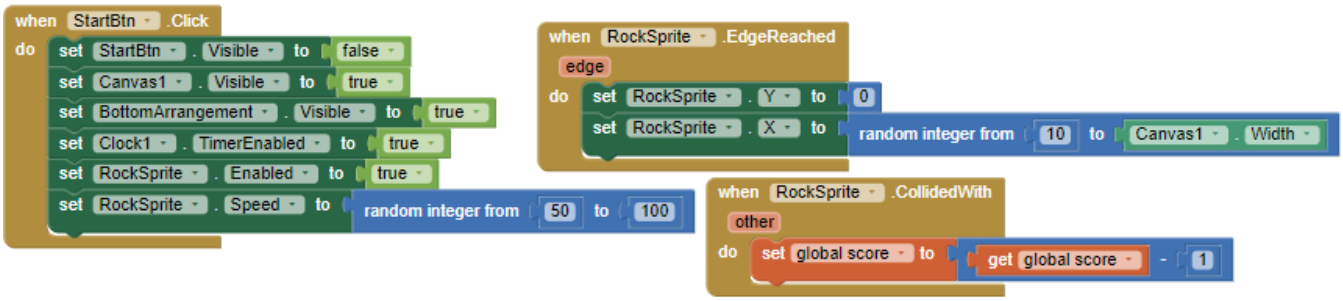
You can calculate how many pieces of fruit the picker fails to catch during a game by subtracting the player's score from the total number of fruits dropped.

In the Blocks Editor, add two new string inputs to the bottom of the **join** block in the **Label1.Text** setter block that displays the score. Fill the first new input with " **- Missed fruit:** " Fill the last input with a **subtraction operator** block from the Math drawer that subtracts the value of the `global score` variable from the value of the `global fruitsDropped` variable.



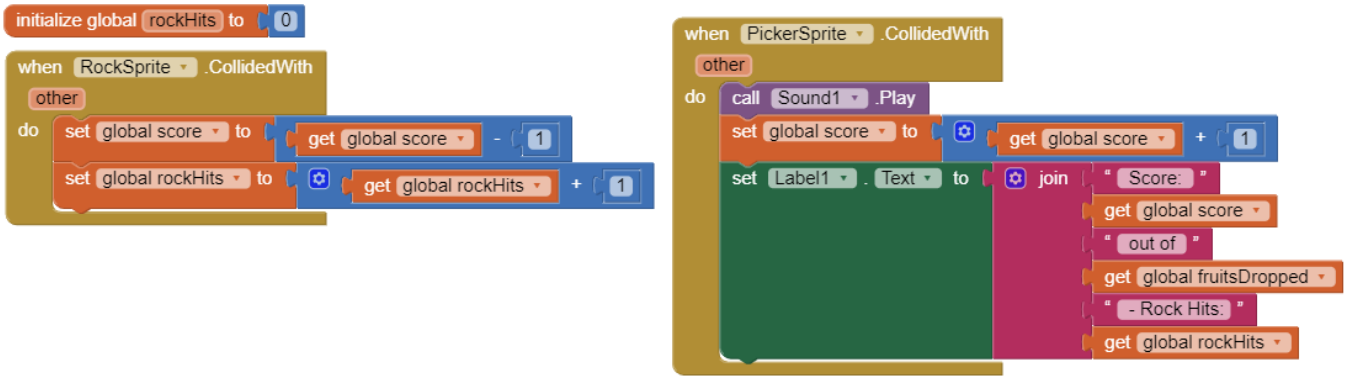
2 – Extend the game so that the frustrated owner of the fruit trees, who can't keep the fruit from falling over the fence, drops rocks down the fence to try to keep the picker from attempting to catch the falling fruit. Reduce the player's score each time the rock hits another sprite. What components and blocks will you add?

a. In the Designer, add another **ImageSprite** to the `Canvas`, and rename it **RockSprite**. In its Properties pane, uncheck **Enabled**; set **Heading** to 270 to make it always move down; set its **Interval** to 250 so that it will move every 250 milliseconds; upload a picture of a rock that is the same width and height as the FruitSprite images and set its **Picture** property to the uploaded picture; and set its **Y** property to 0.
b. In the Blocks Editor, add setter blocks to the **StartBtn Click** event handler to set `RockSprite`'s Enabled property to true and its Speed to a random value between `50` and `100`. Program the **RockSprite EdgeReached** event handler to move `RockSprite` when it drops to the bottom of the `Canvas`, back to the top of the `Canvas` positioned horizontally anywhere from `10` pixels from the `Canvas`'s left edge to the right edge of the `Canvas`. Program the **RockSprite CollidedWith** event handler to subtract one from the value of the `global score` variable when `RockSprite` hits another sprite.



3 – Extend the game even further so that the score label displays the number of times the rock hits another sprite.

In the Blocks Editor, initialize a new global variable, **`rockHits`**, to the value of **`0`**. Add blocks to **`RockSprite`**'s **`CollidedWith`** event handler to add `1` to the value of the `global rockHits` variable when `RockSprite` hits another sprite. Add two new string inputs to the bottom of the **`join`** block in the **`Label1.Text`** setter block that displays the score. Fill the first new input with " **`- Rock Hits:`** " Fill the last input with a **`rockHits`** getter block.

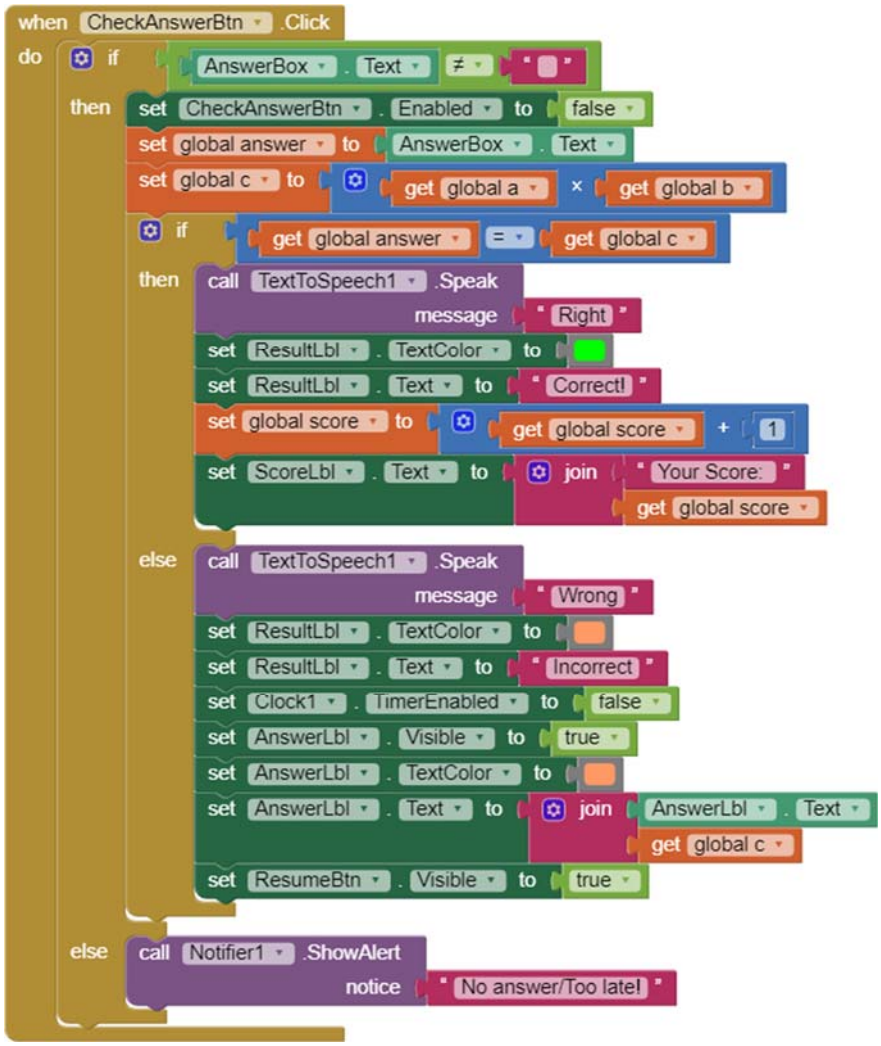


CHAPTER 4 - ON YOUR OWN EXERCISES – POSSIBLE SOLUTIONS

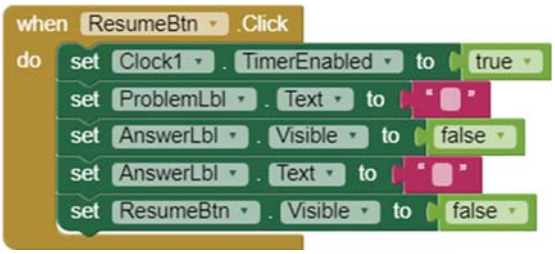
1 – Change the *Multiplication Station* app so that if a user's answer is incorrect, new problems stop appearing and the app displays the correct answer, after which the user may continue the practice session. Where will the app display the correct answer? How will the user resume practice, if desired? Will you need another `if then` block, additional components, or more variables?

a. In the Designer for `Screen2`, add a new **`Label`** from the User Interface drawer right above `ScoreLbl` and rename it **`AnswerLbl`**. In its properties pane, check **FontBold**; set the **Text** to "**`The correct answer is `**"; and uncheck **Visible** so the `Label` doesn't display when the app opens. Add a new **`Button`** from the User Interface drawer right below `AnswerLbl` and rename it **`ResumeBtn`**. In its properties pane, set the **Text** to "**`Keep Practicing`**", and uncheck **Visible** so the `Button` doesn't display when the app opens.

b. In the Blocks Editor for `Screen2`, add 5 setter blocks to the **`CheckAnswerBtn Click`** event handler in the second **`if then else`** block's **`else`** socket to (1) disable the `Clock` Timer; (2) make `AnswerLbl` visible with (3) pink colored text that (4) displays the correct answer (`global c`); and (5) make `ResumeBtn` visible.

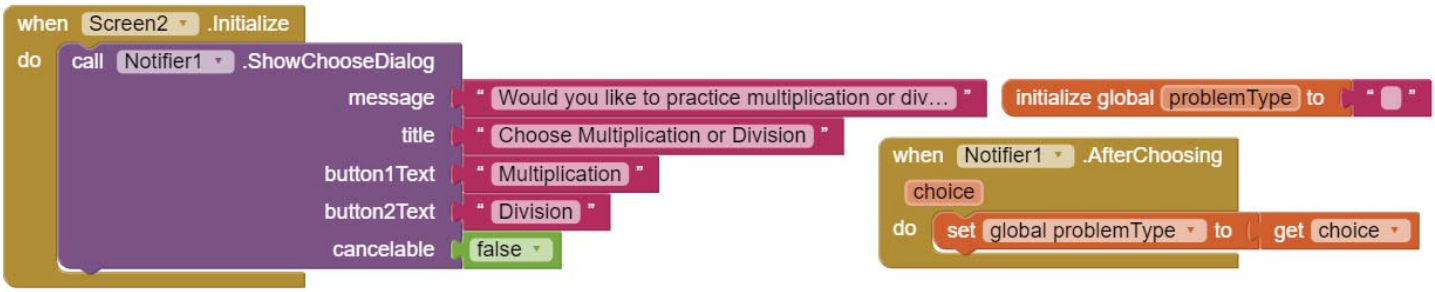


Program the new **ResumeBtn Click** event handler to (1) enable the Clock Timer; (2) clear ProblemLbl's text; (3) make AnswerLbl invisible; (4) clear AnswerLbl's Text; and (5) make ResumeBtn invisible.



2 – Extend the app so that the user can choose to practice multiplication or division. How will your algorithm change? What blocks will you need to add and modify?

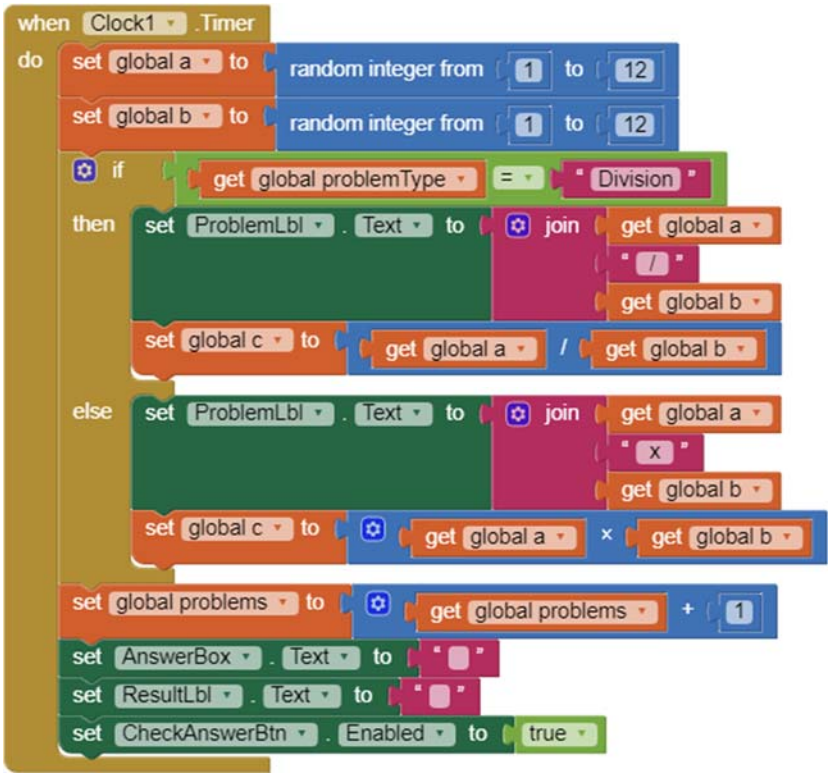
In the Blocks Editor for Screen2, add a **Screen Initialize** event handler that opens Notifier1 to show a dialog to the user so the user can choose to practice multiplication or division. Initialize a new global variable, **problemType**, to the value of an empty string. Program a **Notifier AfterChoosing** event handler to set the value of the problemType variable to the value the user chooses in the Notifier dialog.

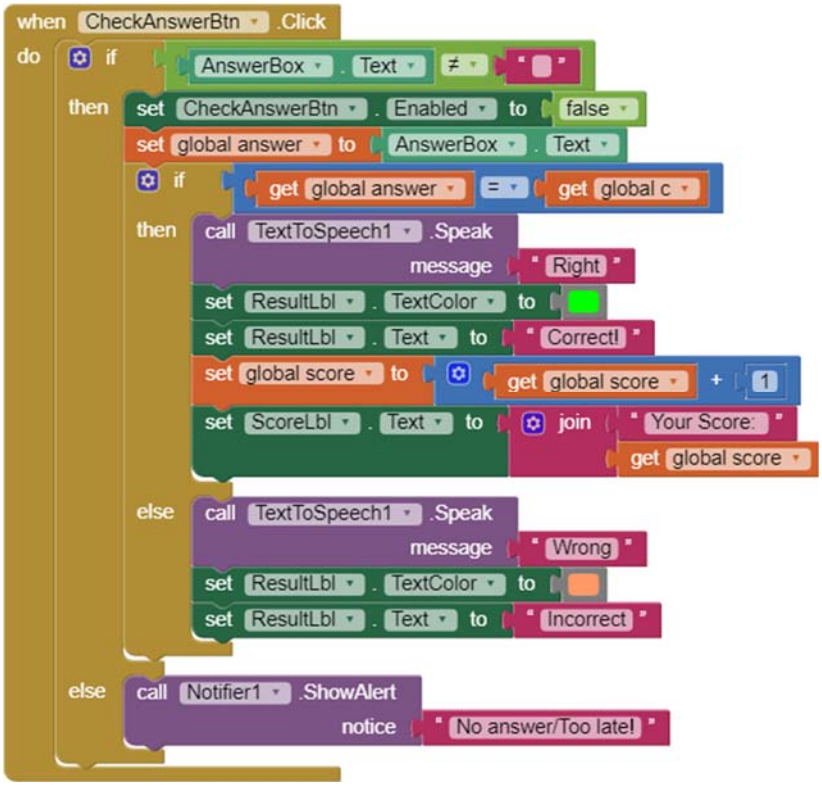


Add an **if then else** block to the **Clock1 Timer** event handler to test whether the value of the problemType variable is "Division," which means the user chose to practice division. If so, set the text in ProblemLbl to display division problems for the user to solve, and set the value of global variable c to global variable a divided by global variable b.

On the other hand, if the value of the problemType variable is not "Division," meaning the user chose to practice multiplication, set the text in ProblemLbl to display multiplication problems for the user to solve, and set the value of global variable c to global variable a multiplied by global variable b.

In the **CheckAnswerBtn Click** event handler, remove from the first **if then else** block's **then** socket the blocks that set the value of global variable c to global variable a multiplied by global variable b.

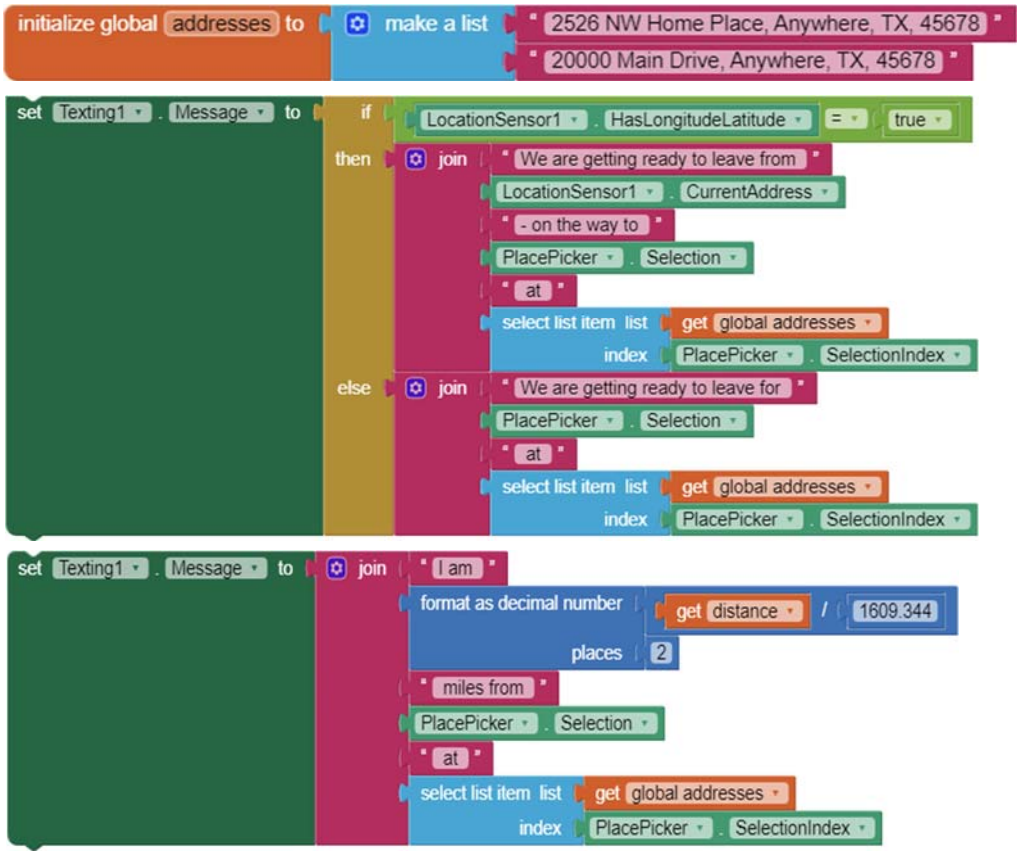

when Clock1 .Timer
do set global a to random integer from 1 to 12
set global b to random integer from 1 to 12
if get global problemType = " Division "
then set ProblemLbl . Text to join get global a
" / "
get global b
set global c to get global a / get global b
else set ProblemLbl . Text to join get global a
" x "
get global b
set global c to get global a × get global b
set global problems to get global problems + 1
set AnswerBox . Text to " "
set ResultLbl . Text to " "
set CheckAnswerBtn . Enabled to true


when CheckAnswerBtn .Click
do if AnswerBox . Text ≠ " "
then set CheckAnswerBtn . Enabled to false
set global answer to AnswerBox . Text
if get global answer = get global c
then call TextToSpeech1 .Speak
message " Right "
set ResultLbl . TextColor to
set ResultLbl . Text to " Correct! "
set global score to get global score + 1
set ScoreLbl . Text to join " Your Score: "
get global score
else call TextToSpeech1 .Speak
message " Wrong "
set ResultLbl . TextColor to
set ResultLbl . Text to " Incorrect "
else call Notifier1 .ShowAlert
notice " No answer/Too late! "

CHAPTER 5 - ON YOUR OWN EXERCISES – POSSIBLE SOLUTIONS

1 – Change the *Beat the Bus* app so that it also provides the destination street address in all text messages sent and retrieves that information from a list variable.

To include the destination street address in the message, in the Blocks Editor, initialize a new global list variable, **addresses**, to hold the addresses for the locations in the global places variable. Add two new string inputs to the two **join** blocks in the **Texting1.Message** setter block in **PlacePicker's AfterPicking** event handler and to the **join** block in the **Texting1.Message** setter block in **LocationSensor1**'s **LocationChanged** event handler. In all three places, fill the first new **join** block input with " **at** ". Fill the second input with a **select list item** block from the Lists blocks drawer that selects the address that is the item at the **PlacePicker SelectionIndex** position in the new global addresses list.



2 – Change the app so that it requires users to enter the destination instead of choosing it from the preset list.

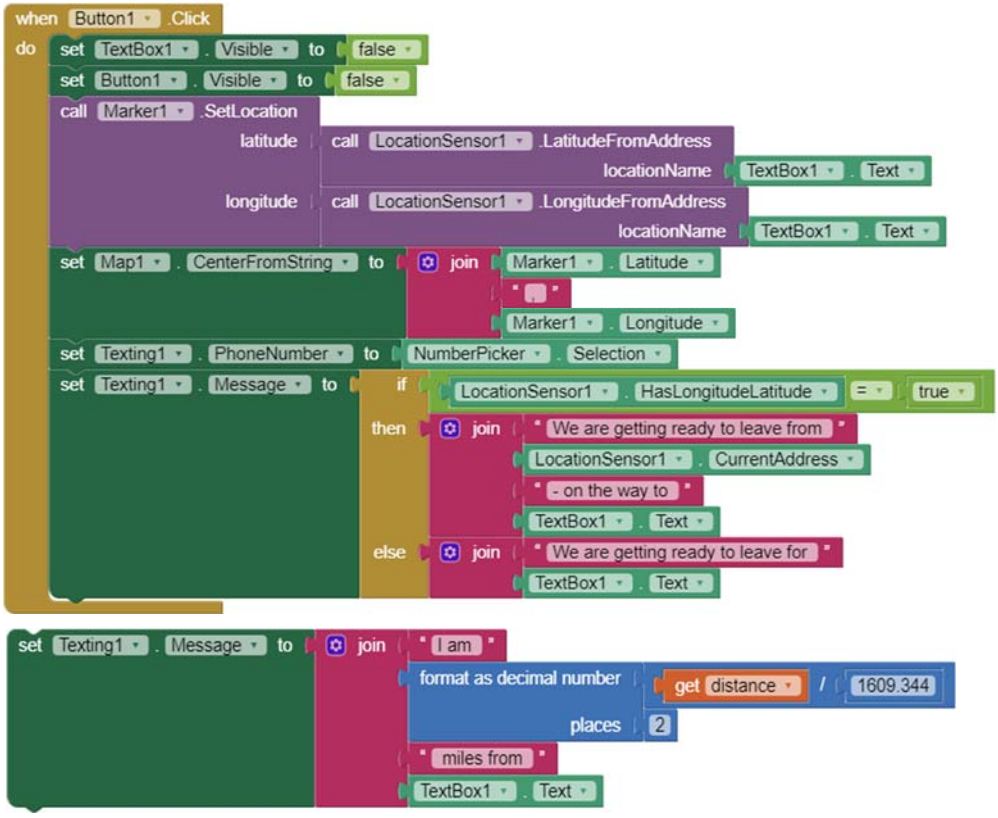
a. In the Designer, under NumberPicker, add a **TextBox** from the User Interface drawer and set its **Hint** property to "**Enter the full destination address**". Add a **Button** from the User Interface drawer beneath the TextBox, and in the Properties pane, check **FontBold**, make the **FontSize** 25, and enter its **Text** as "**Start Tracking**". Also, for both components, make the **Background** yellow and the **Width** Fill parent, and uncheck **Visible** so they don't display when the app opens.

b. In the Blocks Editor, add setters to **NumberPicker**'s **After Picking** event handler to make the new TextBox and Button visible after the user makes a telephone number selection.

Duplicate all blocks in the **PlacePicker**'s **After Picking** event handler and set the copied blocks aside. Delete **PlacePicker.**

Program the new **Button**'s **Click** event handler to hide the TextBox and Button, and then snap in the blocks copied from the **PlacePicker After Picking** event handler. Change those blocks so that the latitude and longitude for **Marker1**'s **SetLocation** method call come from calls to the **LocationSensor**'s **LatitudeFromAddress** and **LongitudeFromAddress** methods using the text (address) entered into the TextBox (**TextBox1.Text**) as the **locationName**.

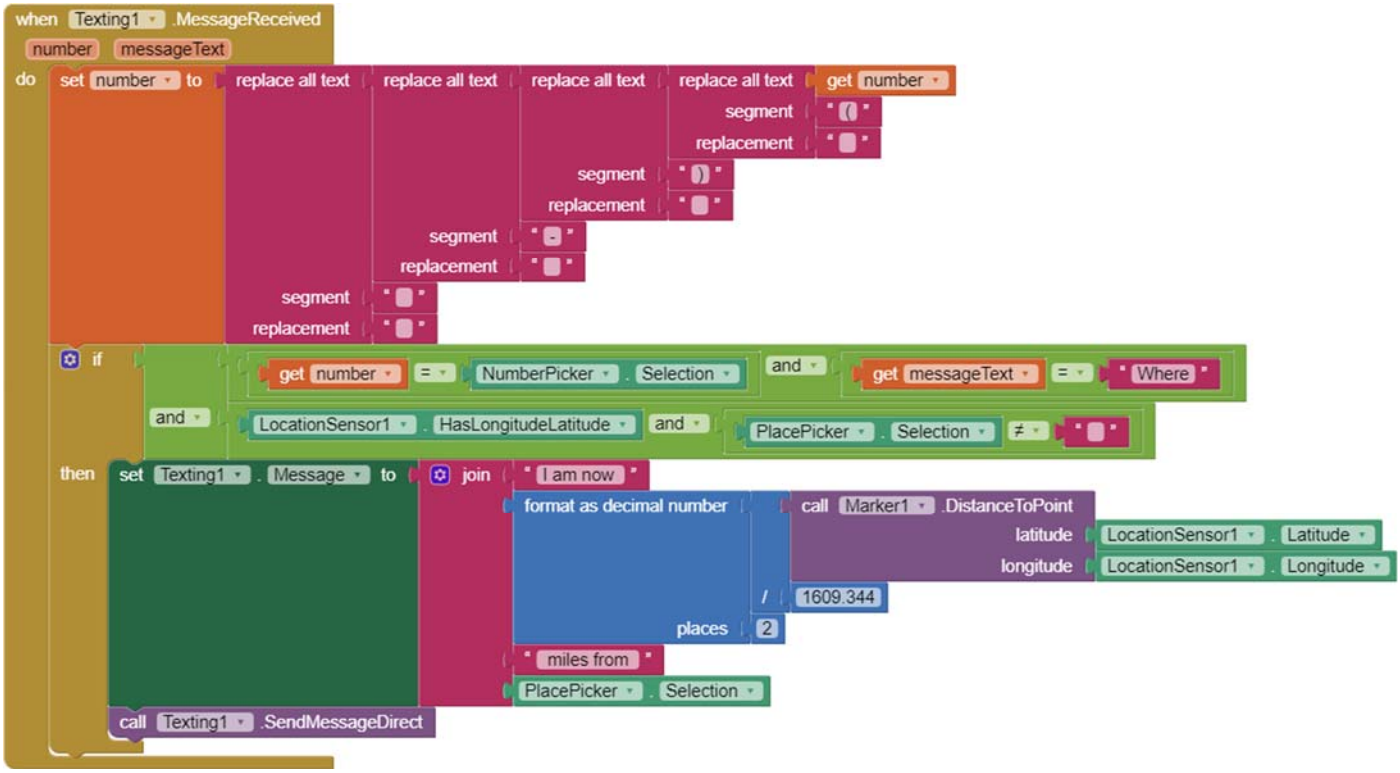
In the **Texting1 Message** setter blocks in both the new **Button Click** event handler and in **LocationSensor1**'s **LocationChanged** event handler, drag in **TextBox1.Text** getter blocks to indicate the user's destination.



3 – Extend the app for Android devices so that the app responds to specific text messages from the selected number by texting the user's current location at that time. How will your algorithm change? What components and blocks will you need to add and modify?

The blocks below accomplish this task by programming the app to send a text message with location information to the selected phone number only if all of the following four conditions are met: (a) the incoming phone `number` is identical to the selected phone number; (b) the message received from the incoming phone `number` consists only of the word "Where;" (c) `LocationSensor1` has retrieved the app's current latitude and longitude; and (d) the user has selected a destination.

Note that we start the code by changing the value of the incoming `number` using a series of **`replace all`** blocks from the Text blocks drawer to strip all nonnumeric characters (parentheses, dashes and spaces) from that incoming phone `number`. We strip away these characters so that the incoming `number` is left with numbers only, just like the user's selected telephone number from our global `phoneNumbers` variable. This makes the incoming phone `number` easier to compare with the selected phone number in the following **`if then`** block.



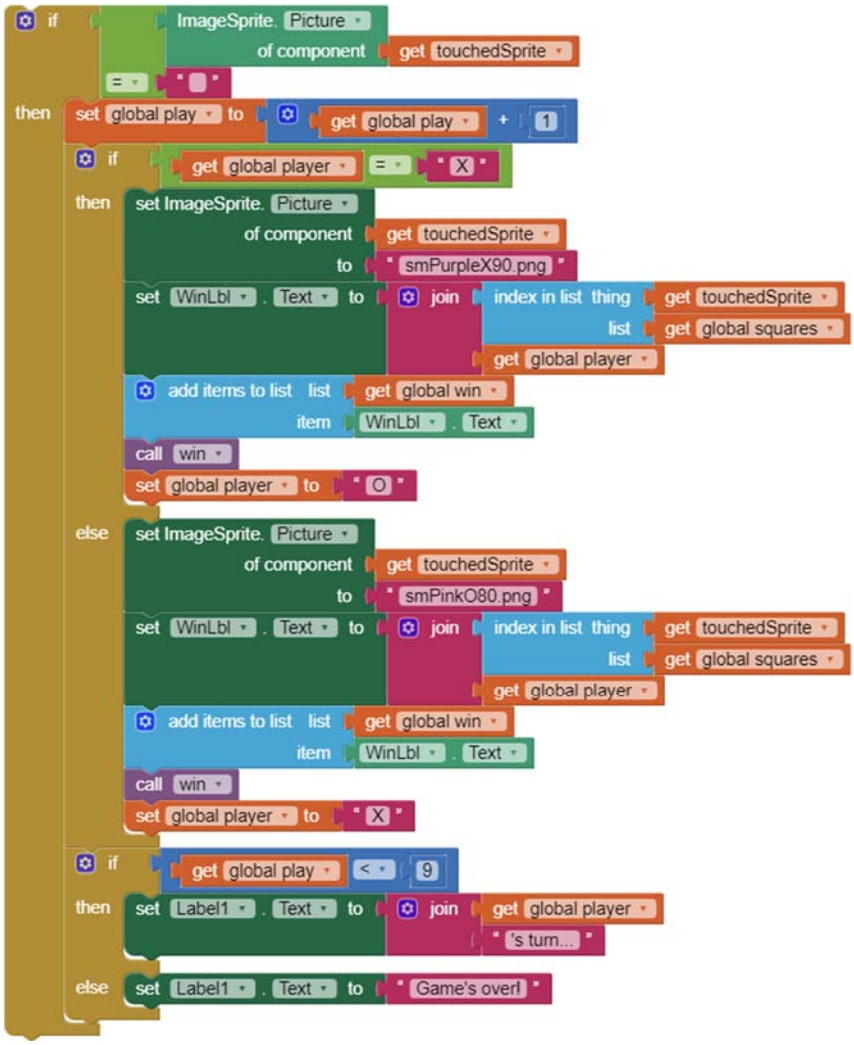
CHAPTER 6 - ON YOUR OWN EXERCISES – POSSIBLE SOLUTIONS

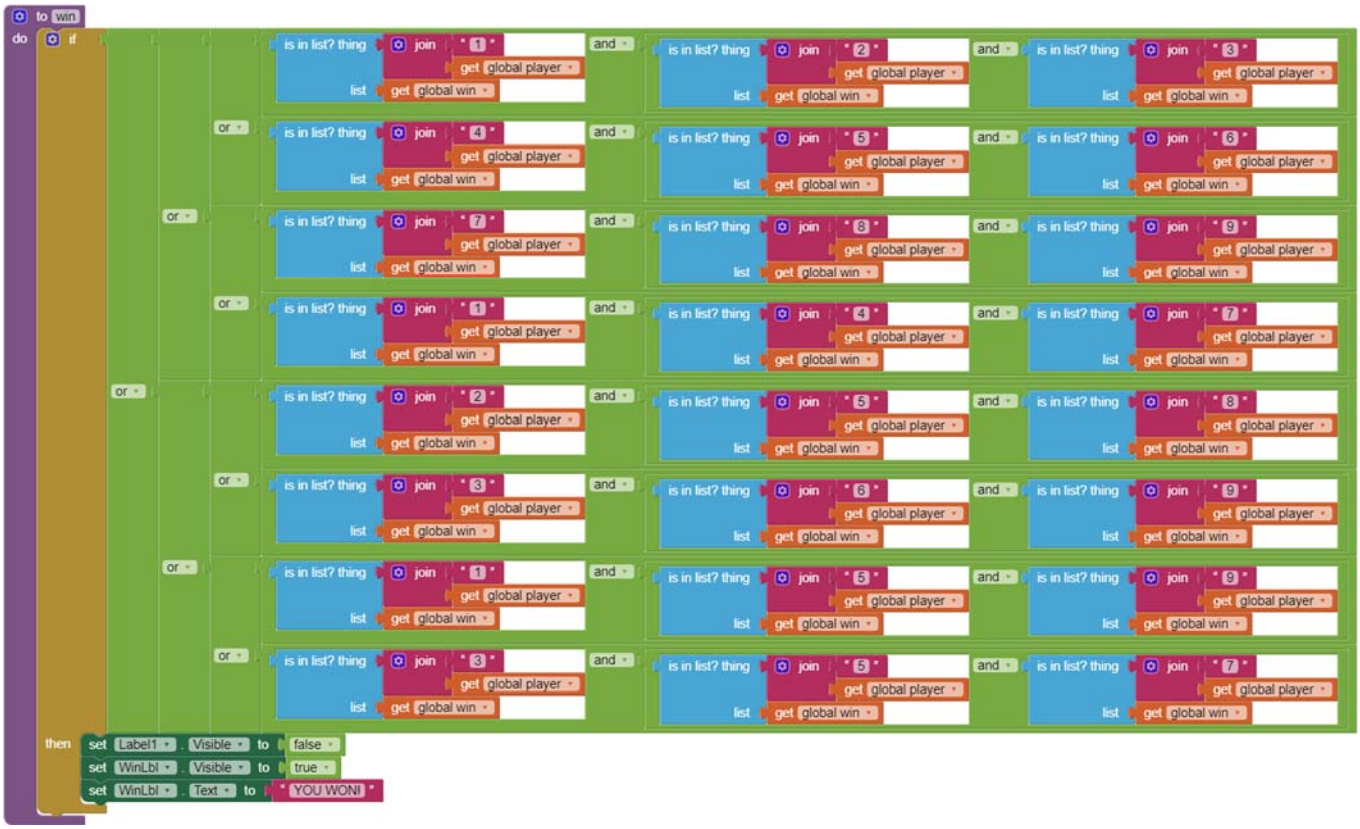
1 – Extend the *Tic Tac Toe* app so that it indicates when a player has won, meaning a player has placed X's or O's in three squares in a row, either horizontally, vertically, or diagonally.

a. In the Designer, add another **Label** from the User Interface drawer above Button1, so the app will have a place to display the winner, and rename it **WinLbl**. In the Properties pane, make its **FontBold** and **FontSize** 25 and uncheck **Visible** so the Label doesn't display until we program it to show.

b. In the Blocks Editor, create another global list variable, **win**, and initiate it to an empty list. Add code to both the **then** and **else** sockets of the first **if then else** block within the **in** socket of the **Canvas TouchDown** event handler. This new code will set WinLbl's Text to the index number of the touchedSprite in the global squares list joined with the value of the global player variable, and then add WinLbl's Text to the new global win list variable. These blocks store each touched ImageSprite number and the player who touched it in the global win list variable.

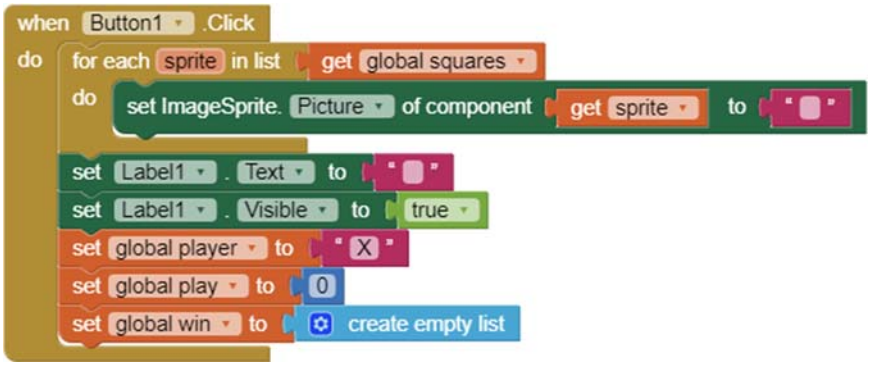
Also, in the code to the right, you'll see two **call win** blocks, which are calls to a new **win** procedure. Creating a procedure here eliminates the need to copy and paste the procedure's multiple lines of code twice, each place where you see the **call win** block. You will learn how to create procedures in the next chapter. In the meantime, you will need to replace each **call win** block to the right with the full **if then** block contained in the **win** procedure on the next page.





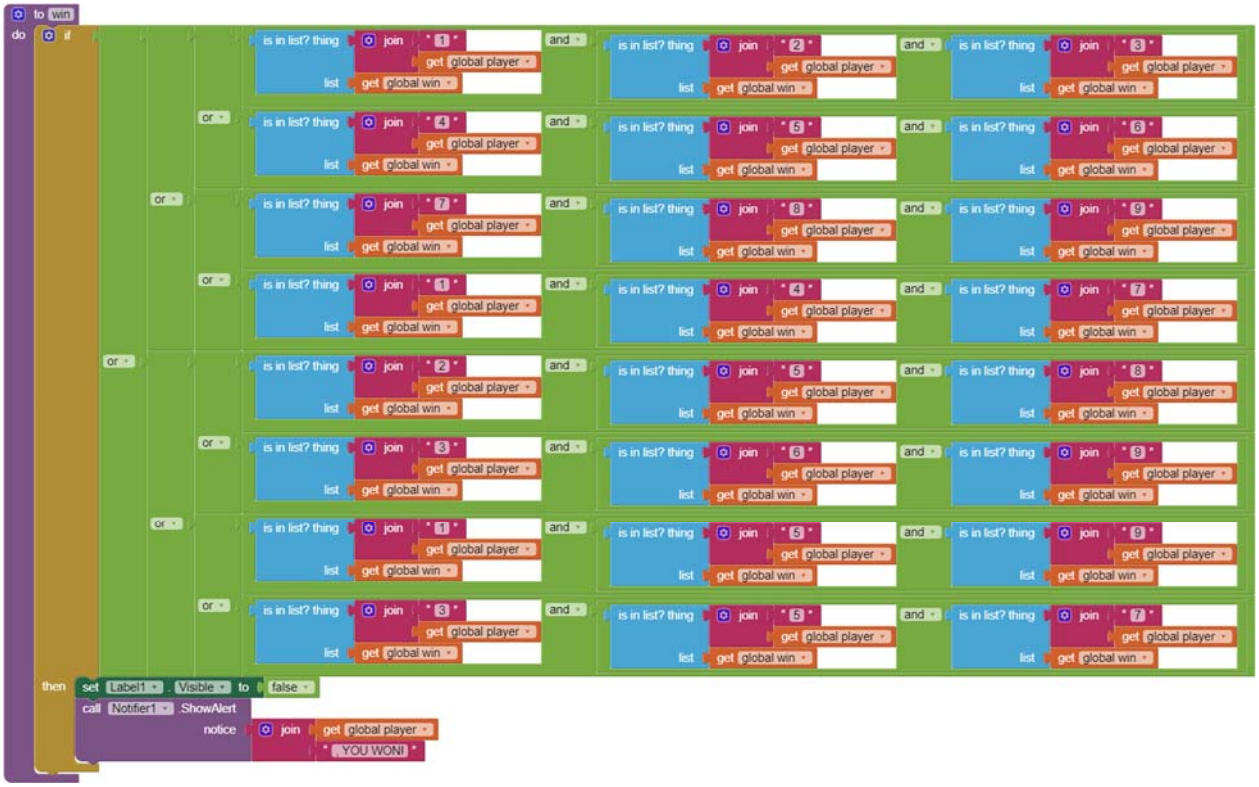
The **win** procedure blocks above evaluate whether a player has won the game, meaning the player has touched three squares in a row, either horizontally, vertically, or diagonally. Each time a player touches an empty board square, the blocks look into the global `win` list variable to see if that player has touched the three `ImageSprites` in one of the eight winning series of board squares: squares 1, 2 and 3; squares 4, 5 and 6; squares 7, 8 and 9; squares 1, 4 and 7; squares 2, 5 and 8; squares 3, 6 and 9; squares 1, 5 and 9; or squares 3, 5 and 9. If so, the code hides `Label1` (which is the `Label` that displays either who's turn it is to play or "Game's Over!") and instead shows `WinLbl` displaying "YOU WON!"

Change the **Button1** (reset button) **Click** event handler to add setter blocks that make `Label1` visible again and that empty the global `win` list variable when a player clicks `Button1`.



2 – Extend the app further so that it uses an additional component to indicate which player has won the game.

a. In the Designer, add a `Notifier` component from the User Interface drawer, and make its **BackgroundColor** Magenta, **NotifierLength** Short and **TextColor** Black.
b. In the Blocks Editor, change the code so that `WinLbl` stays hidden and the `Notifier` indicates which player has won the game.



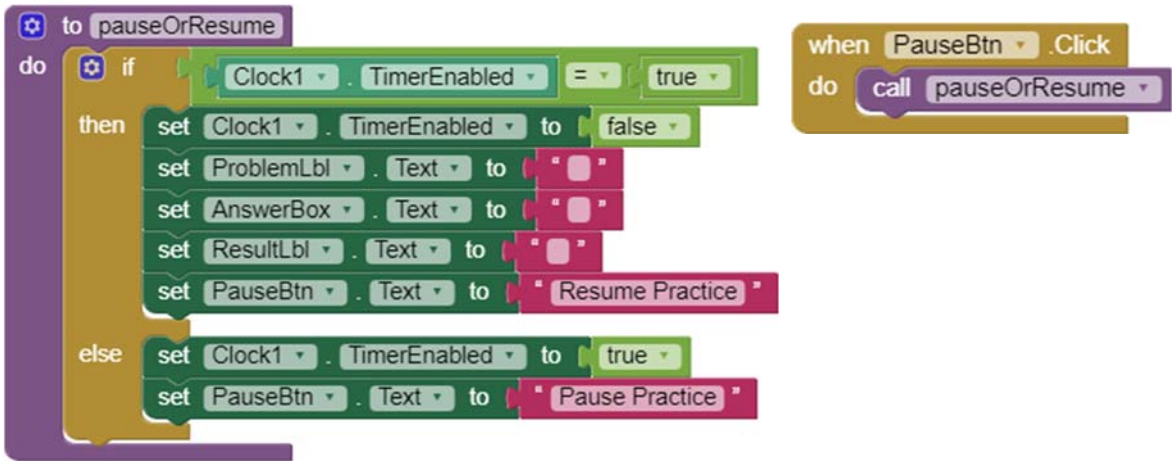
CHAPTER 7 - ON YOUR OWN EXERCISES – POSSIBLE SOLUTIONS

1 – In *Multiplication Station II*, try adding a pause button to the practice screen so users can temporarily stop practicing when they need to. Write and call a procedure as part of this extension.

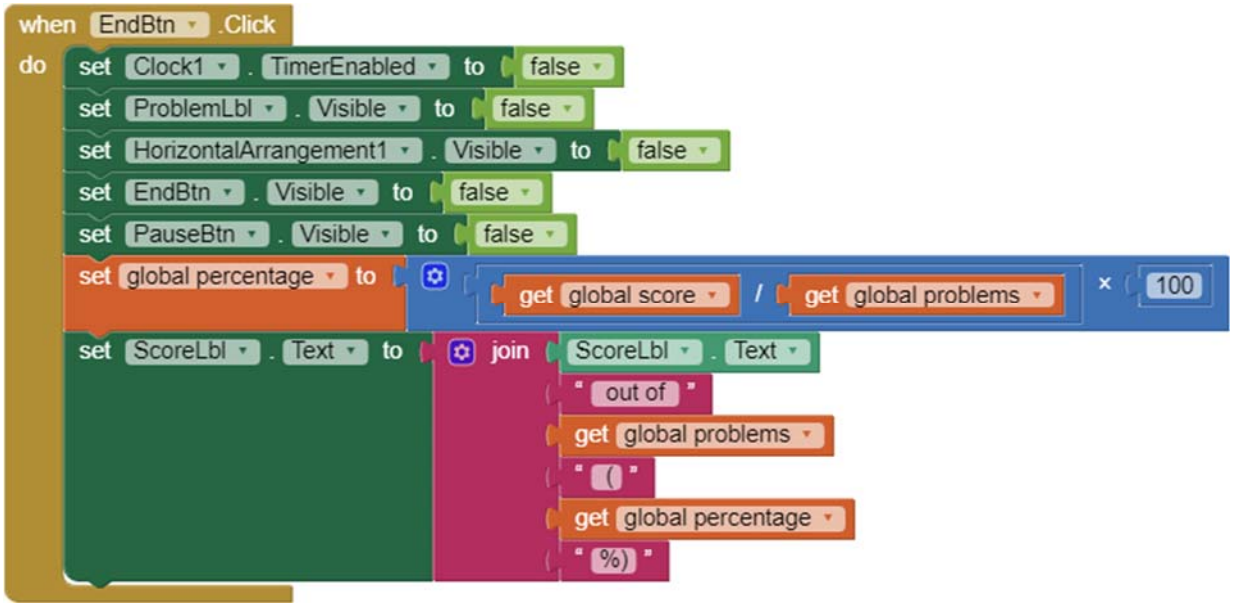
a. In the Designer for Screen2, add a new **Button** from the User Interface drawer right above `ScoreLbl` and rename it **PauseBtn**. In its properties pane, set the **Text** to **Pause Practice**.

b. In the Blocks Editor for Screen2, create the **pauseOrResume** procedure to check whether the `Clock` Timer is enabled, meaning the user is currently practicing. If the Timer is enabled, stop the Timer, remove all characters from `ProblemLbl`, `AnswerBox`, and `ResultLbl`, and change the `PauseBtn` Text to `Resume Practice`. On the other hand, if the `Clock` Timer is not enabled, meaning the user has already paused practice, enable the Timer and reset the `PauseBtn` Text to `Pause Practice`.

   Program the **PauseBtn Click** event handler to call the new **pauseOrResume** procedure.

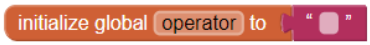


   Add a setter block to the **EndBtn Click** event handler to hide `PauseBtn` when the user clicks `EndBtn`.

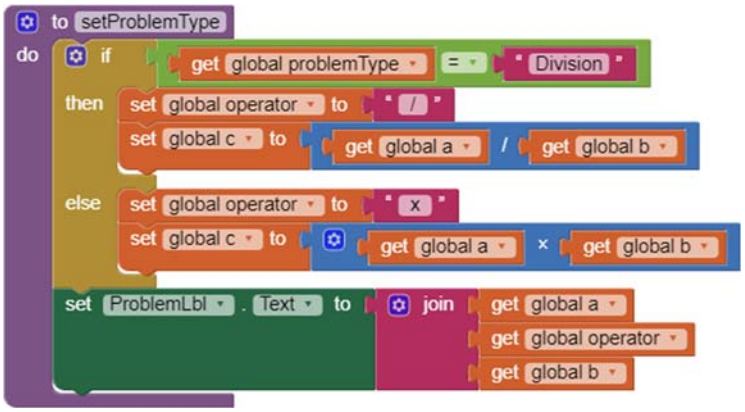


2 – Write and call a procedure that refactors the code you wrote in Chapter 4 for the first *Multiplication Station* app to allow users to choose to practice multiplication or division problems.

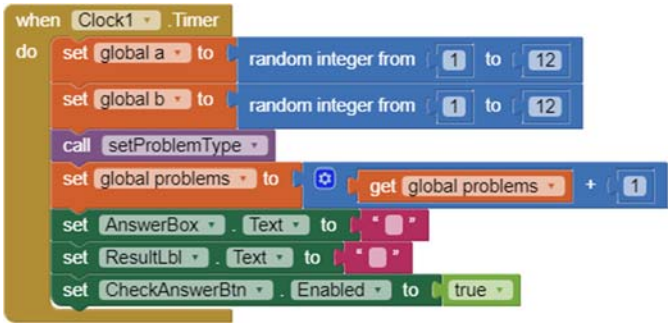
In the Blocks Editor for Screen2, initialize a new global **operator** variable, to the value of an empty string.



Create the **setProblemType** procedure (a) to check for the value of the global problemType variable. If the value is Division, then set the value of global variable operator to / (the division operator) and set the value of global variable c to global variable a divided by global variable b. If the value of the problemType variable is not Division, then set the value of global variable operator to x (the multiplication operator) and set the value of global variable c to global variable a multiplied by global variable b; and (b) to set the Text in ProblemLbl to display the joined values of global variable a, global variable operator, and global variable b.



Replace the entire **if then else** block in the **Clock Timer** event handler with a **call setProblemType** block.



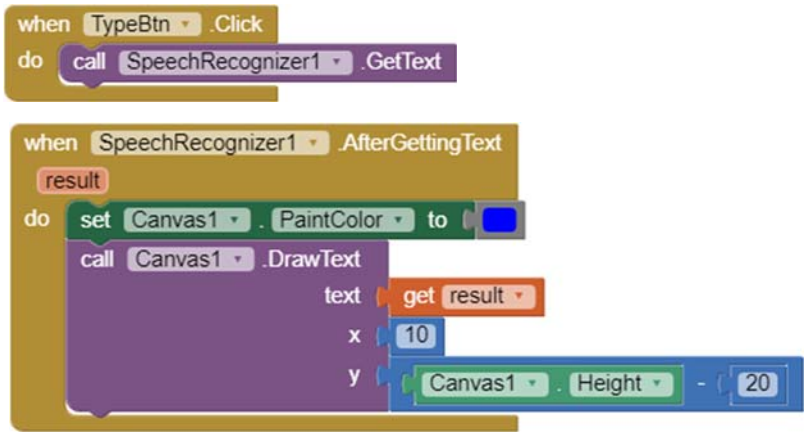
CHAPTER 8 - ON YOUR OWN EXERCISES – POSSIBLE SOLUTIONS

1 – Change *Virtual Shades* so that it allows users to choose the Canvas paint color to draw and type on the Canvas.

a. In the Designer, add a **Spinner** from the User Interface drawer at the bottom of VerticalArrangement1. In the Properties pane, enter **Choose a paint color,red,blue,green,black** for its **ElementsFromString** property and make its **Width** Fill parent.
b. In the Blocks Editor, create a new global list variable, **colors**, to correspond with Spinner1's elements. Note that the first item in the colors list is set to the default color black, in case the user does not select a color from the Spinner. Change the **Canvas1 PaintColor** setter blocks in both the **Canvas1 Dragged** and **Notifier1 AfterTextInput** event handlers to select the color that is at the Spinner1 SelectionIndex position in the colors list.

2 – Change the app so that it allows users to speak the words they want entered on the Canvas.

a. In the Designer, add a **SpeechRecognizer** component from the Media drawer.
b. In the Blocks Editor, change the **TypeBtn Click** event handler to call the **SpeechRecognizer1 GetText** method. Program the **SpeechRecognizer1 AfterGettingText** event handler to set the paint color to blue and draw the spoken text on the Canvas.



3 – Extend the app so that users can opt not to take a background photo and instead create art on a blank Canvas.

a. In the Designer, add a **CheckBox** from the User Interface drawer beneath TakePicBtn. Change its **Background** to Red, check **FontBold**, make the **FontSize** 20, change **Width** to Fill parent, and set the **Text** to "OR Click here to skip the pic".
b. In the Blocks Editor, create the **openCanvas** procedure to hide the TakePicBtn and new CheckBox and show VerticalArrangement1, which holds the Canvas and other buttons. Program the **CheckBox Changed** event handler to call the **openCanvas** procedure. Modify the **Camera AfterPicture** event handler to call the **openCanvas** procedure after setting the Canvas background to the image taken by the Camera.