CHAPTER 4

Decision Blocks and Boolean Logic

GLOBAL EDITION
Starting Out with App Inventor for Android
Tony Gaddis • Rebecca Halsey
ALWAYS LEARNING
PEARSON

Topics

- Introduction to Decision Blocks
- Relational Operators and the `if` Block
- The `if then else` block
- A First Look at Comparing Strings
- Logical Operators
- Nested Decision Blocks
- The `if then else if` block
- Working with Random Numbers
- The `Screen's Initialize` Event
- The `ListPicker` Component
- The `Checkbox` Component

Introduction to Decision Blocks

- Sometimes a program needs to ”decide” whether or not to execute certain instructions. App Inventor provides three blocks for making decisions.
- So far you have worked with number values and text (string) values.

Introduction to Decision Blocks

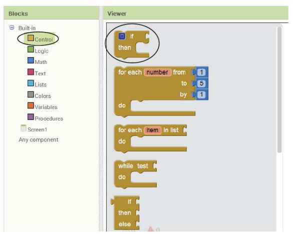
- Programs can also work with the values `true` and `false`.
- `true` and `false`, are known as *Boolean* values.
- `true` and `false` values are commonly used in decision making.

Introduction to Decision Blocks

The `if then` Block

App Inventor provides the `if then` block for making decisions.

Figure 4-1 The Decision Blocks (*Source:* MIT App Inventor 2)



Introduction to Decision Blocks

The `if then` Block

In Figure 4-2, notice that the `if then block` has two sockets: one for the `if` part, one for the `then` part.

Figure 4-2 The `if then` Block (*Source:* MIT App Inventor 2)

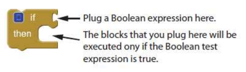
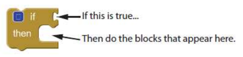

if
then
Plug a Boolean expression here.
The blocks that you plug here will be executed ony if the Boolean test expression is true.

Figure 4-3 How to Think About the `if then` Block (*Source:* MIT App Inventor 2)

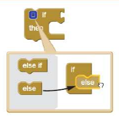


Introduction to Decision Blocks

The `if then else` Block

- The `if then` block is a mutator block.
- A mutator block has the ability to change in some way.
- Click the blue box that appears in the upper-left corner and the mutator bubble will appear.
- You can change the `if then` block to an `if then else` block.

Figure 4-5 Changing the `if then` Block to an `if then else` Block (*Source:* MIT App Inventor 2)

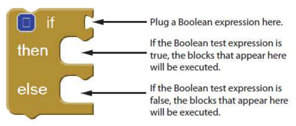


Introduction to Decision Blocks

The `if then else` Block

If the *Boolean* expression is `true`, the instructions that appear in the `then` sockets will be executed. If the *Boolean* expression is `false`, instructions that appear in the `else` socket will be executed.

Figure 4-6 The `if then else` Block (*Source:* MIT App Inventor 2)

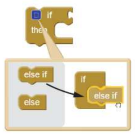

if
then
else
Plug a Boolean expression here.
If the Boolean test expression is true, the blocks that appear here will be executed.
If the Boolean test expression is false, the blocks that appear here will be executed.

Introduction to Decision Blocks

The `if then else if` Block

Use the `if then` blocks mutator bubble to change the block into an `if then else if` block.

Figure 4-8 Changing an `if then` Block to an `if then else if` Block (*Source:* MIT App Inventor 2)

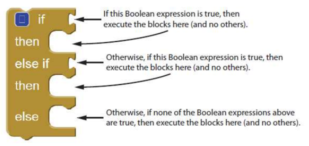


Introduction to Decision Blocks

The `if then else if` Block

- Drag the `else` block from the left side then insert it on the right side of the bubble.
- This creates an `if then else if` block like the one shown in Figure 4-10.

Figure 4-10 An `if then else if` Block (*Source:* MIT App Inventor 2)

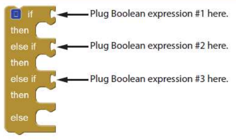

if
then
else if
then
else
If this Boolean expression is true, then execute the blocks here (and no others).
Otherwise, if this Boolean expression is true, then execute the blocks here (and no others).
Otherwise, if none of the Boolean expressions above are true, then execute the blocks here (and no others).

Introduction to Decision Blocks

The `if then else if` Block

Use the mutator bubble to add as many `else if` sections as you need.

Figure 4-11 An `if then else if` Block that Can Test Three Boolean Expressions (*Source:* MIT App Inventor 2)

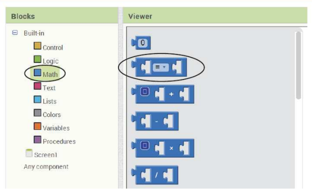

if
then
else if
then
else if
then
else
Plug Boolean expression #1 here.
Plug Boolean expression #2 here.
Plug Boolean expression #3 here.

Relational Operators and the `if` Block

A *relational operator* determines whether a specific relationship exists between two values.

Access the relational operators in the *Math* drawer.

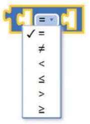
Figure 4-12 The Relational Operator Blocks (*Source:* MIT App Inventor 2)



Relational Operators and the `if` Block

Change it to any other relational operator by clicking on the down arrow (▼) .

Figure 4-13 The Relational Operator Dropdown Menu (*Source:* MIT App Inventor 2)



Relational Operators and the `if` Block

Table 4-1 summarizes each of the relational operator blocks.

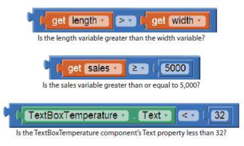
Table 4-1 The Relational Operator Blocks (*Source:* Pearson Education, Inc.)

Operator Block	Description
>	This is the *greater than operator*. It returns `true` if the operand on the left is greater than the operand on the right. Otherwise, it returns `false`.
≥	This is the *greater than or equal to operator*. It returns `true` if the operand on the left is greater than or equal to the operand on the right. Otherwise, it returns `false`.
<	This is the *less than operator*. It returns `true` if the operand on the left is less than the operand on the right. Otherwise, it returns `false`.
≤	This is the *less than or equal to operator*. It returns `true` if the operand on the left is less than or equal to the operand on the right. Otherwise, it returns `false`.
=	This is the *equal to operator*. It returns `true` if the operand on the left is equal to the operand on the right. Otherwise, it returns `false`.
≠	This is the *not equal to operator*. It returns `true` if the operand on the left is not equal to the operand on the right. Otherwise, it returns `false`.

Relational Operators and the `if` Block

Figure 4-14 shows some examples of relational operators with operands plugged in.

Figure 4-14 Relational Operator Block Examples (*Source:* MIT App Inventor 2)


get length > get width
Is the length variable greater than the width variable?
get sales ≥ 5000
Is the sales variable greater than or equal to 5,000?
TextBoxTemperature . Text < 32
Is the TextBoxTemperature component's Text property less than 32?

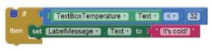
Relational Operators and the `if` Block

- The top example determines whether the `length` variable is greater than the `width` variable.
- The middle example determines whether the `sales` variable is greater than or equal to 5,000.
- The bottom example determines whether the `TextBox` temperature component's `Text` property contains a value that is less than 32.0.

Relational Operators and the `if` Block

- Figure 4-15 shows an example of a complete `if then` block.

Figure 4-15 Example `if then` Block (*Source:* MIT App Inventor 2)

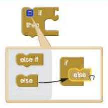


PEARSON

The `if then else` Block

- An `if then else` block will execute one set of blocks if it's Boolean expression is true or another set of blocks if its Boolean expression is false.
- Use the `if then` block's mutator bubble to convert the block into an `if then else` block.

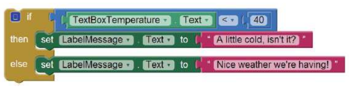
Figure 4-32 Changing the `if then` Block to an `if then else` Block (*Source:* MIT App Inventor 2)



PEARSON

The `if then else` Block

Figure 4-34 Example of the `if then else` Block (*Source:* MIT App Inventor 2)


if
TextBoxTemperature . Text
<
40
then
set LabelMessage . Text to
" A little cold, isn't it? "
else
set LabelMessage . Text to
" Nice weather we're having! "

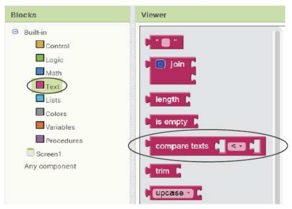
- The Boolean test expression uses the less than (<) operator to determine whether `TextBoxTemperature.Text` is less than 40.0.
- If the Boolean expression is true, then the text *A little cold, isn't it?* is assigned to `LabelMessage.Text`
- If the Boolean expression is false, the text *Nice weather we're having!* is assigned to `LabelMessage.Text.`

PEARSON

A First Look At Comparing Strings

- The `compare texts` block compares to strings.
- `compare texts`, found in the *Text* drawer of the *Built-in* section of the *Blocks* column as shown in Figure 4-50.

Figure 4-50 The `compare texts` Block (*Source:* MIT App Inventor 2)

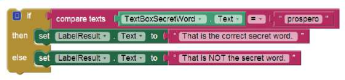


PEARSON

A First Look At Comparing Strings

- The `compare texts` Block let's you compare two strings and determine whether one string is alphabetically less than, greater than, or equal to another string.
- `compare texts =` operator has to sockets. The operator returns true if they are equal. Otherwise it returns false.

Figure 4-52 Comparing Two Strings (*Source:* MIT App Inventor 2)



```
if    compare texts  TextBoxSecretWord . Text  =  " prospero "
then  set LabelResult . Text to " That is the correct secret word. "
else  set LabelResult . Text to " That is NOT the secret word. "
```

Logical Operators

- The logical `and` operator and the logical `or` operator allow you to connect multiple Boolean expressions to create a compound expression.
- Logical operators are used to create complex Boolean expressions.

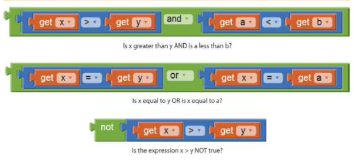
Table 4-4 Logical Operator Blocks (*Source:* Pearson Education, Inc.)

Operator Block	Description
and	This is the logical `and` operator. It has sockets for two Boolean expressions. The `and` block returns `true` if both of the Boolean expressions are `true`, or `false` otherwise.
or	This is the logical `or` operator. It has sockets for two Boolean expressions. The `or` block returns `true` if either of the Boolean expressions are `true`. If both of the Boolean expressions are `false`, the `or` block returns `false`.
not	This is the logical `not` operator. It is a unary operator, meaning it works with only one operand (you can plug only one Boolean expression into this block). The `not` operator reverses the truth of the expression that is plugged into it. If it is applied to an expression that is `true`, the operator returns `false`. If it is applied to an expression that is `false`, the operator returns `true`.

Logical Operators

Figure 4-53 shows some examples of using the logical operator blocks.

Figure 4-53 Logical Operator Block Examples (*Source:* MIT App Inventor 2)



```
get x > get y  and  get a < get b
```

Is x greater than y AND is a less than b?

```
get x = get y  or  get x = get a
```

Is x equal to y OR is x equal to a?

```
not  get x > get y
```

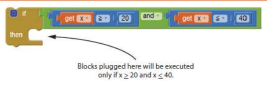
Is the expression x > y NOT true?

Logical Operators

Checking Numeric Ranges with Logical Operators

- When determining whether a number is inside a range, it is best to use the `and` operator.
- The compound Boolean expression being tested by the `if` block in Figure 4-54 is true only when `x` is greater than or equal to 20 and less than or equal to 40.

Figure 4-54 Determining Whether a Number is Inside a Numeric Range (*Source:* MIT App Inventor 2)



```
if    get x ≥ 20  and  get x ≤ 40
then
```

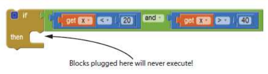
Blocks plugged here will be executed only if x ≥ 20 and x ≤ 40.

Logical Operators

Checking Numeric Ranges with Logical Operators

- When determining whether a number is outside a range, it is best to use the `or` operator.
- The compound Boolean expression shown in Figure 4-56 would never test true as `x` cannot be less than 20 and at the same time be greater than 40.

Figure 4-56 Logic Error (*Source:* MIT App Inventor 2)



```
if   get x < 20  and  get x > 40
then
```

Nested Decision Blocks

A *nested* decision block is written inside the `then` or `else` section of another decision block.

For example in Tutorial 4-5 you will create an app that reads a test score and displays a grade.

The logic of determining the grade can be expressed like this:

If the test score is less than 60, then the grade is F.

Else, if the test score is less than 70, then the grade is D.

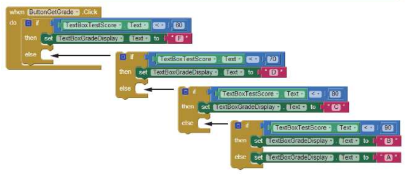
Else, if the test score is less than 80, then the grade is C.

Else, if the test score is less than 90, then the grade is B.

Else, the grade is A .

Nested Decision Blocks

Figure 4-61 Assembling the Nested Decision Blocks in the Grader App (*Source:* MIT App Inventor 2)



```
when ButtonGetGrade .Click
do  if   TextBoxTestScore . Text < 60
    then set TextBoxGradeDisplay . Text to " F "
    else
        if   TextBoxTestScore . Text < 70
        then set TextBoxGradeDisplay . Text to " D "
        else
            if   TextBoxTestScore . Text < 80
            then set TextBoxGradeDisplay . Text to " C "
            else
                if   TextBoxTestScore . Text < 90
                then set TextBoxGradeDisplay . Text to " B "
                else set TextBoxGradeDisplay . Text to " A "
```

The `if then else if` Block

The `if then else if block` tests a series of conditions. It is often simpler to test a series of conditions with the `if then else if` block then with a set of nested `if then else` blocks.

The `if then else if` Block

- Use the `if then` block's mutator bubble to convert the block into an `if then else` block as in Figure 4-65.
- Click and drag the `else if block` from the left side of the bubble and insert it on the right side of the bubble.
- Drag the `else` block from the left side of the and insert it into the right side of the bubble as in Figure 4-66.

The `if then else if` Block

Figure 4-65 Changing an `if then` Block to an `if then else if` Block
(*Source:* MIT App Inventor 2)

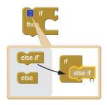
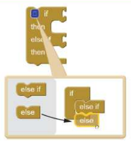


Figure 4-66 Adding an `else` Section to an `if then else if` Block
(*Source:* MIT App Inventor 2)



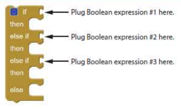
The `if then else if` Block

- When the `if then else if` block executes, Boolean expression #1 is tested.
- If the Boolean expression #1 is true, the instructions in the `then` socket that immediately follow are executed and the rest of the block is ignored.
- If Boolean expression #1 is false, however, the program jumps to the very next `else if` section and tests Boolean expression number two.

The `if then else if` Block

Use the mutator bubble to add as many else if sections as you need as in Figure 4-68.

Figure 4-68 An `if then else if` Block that Can Test Three Boolean Expressions
(*Source:* MIT App Inventor 2)


if
then
else if
then
else if
then
else
Plug Boolean expression #1 here.
Plug Boolean expression #2 here.
Plug Boolean expression #3 here.

Working with Random Numbers

Random numbers are useful for lots of different programming tasks. Random numbers are:

- Commonly used in games such as rolling dice or cards.
- In simulation programs. In some simulations, the computer must randomly decide how some living being will behave.
- Useful in statistical programs that must randomly select data for analysis.
- Commonly used in computer security to encrypt sensitive data.

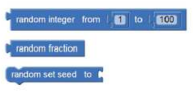
Working with Random Numbers

App Inventor provides blocks for generating random numbers as shown in Figure 4-70. Here is a summary of each block:

- The `random integer` block is a function that takes two arguments: `from` and `to`.
- The `random fraction` block is a function that returns a random fractional number between zero and one.
- The `random set seed` function let's you specify a seed value for random number generation.

Working with Random Numbers

Figure 4-70 The Random Number Blocks (Source: MIT App Inventor 2)

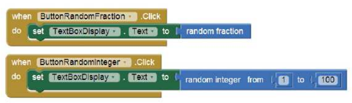


Working with Random Numbers

- Figure 4-71 shows the screens for the `RandomNumberDemo` project. When the user clicks the *Generate Random Fraction* button, the app displays a random fraction between 0 and 1.
- When the user clicks the *Generate Random Integer* button, the app displays a random integer within the range of 1 and 100.
- Figure 4-72 shows the `Click` event handlers and Figure 4-73 shows the app running in the emulator.

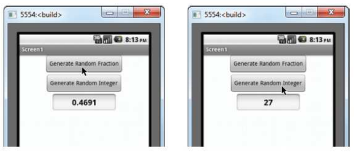
Working with Random Numbers

Figure 4-72 The `Click` Event Handlers (Source: MIT App Inventor 2)



```
when ButtonRandomFraction .Click
do  set TextBoxDisplay . Text to random fraction

when ButtonRandomInteger .Click
do  set TextBoxDisplay . Text to random integer from 1 to 100
```

Figure 4-73 The App Running in the Emulator (Source: MIT App Inventor 2)



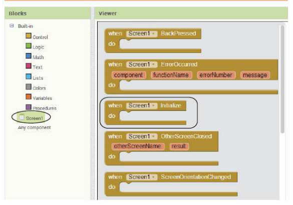
The `Screen's` `Initialize` Event

- If you need to perform set up operations when the app starts, you can create an event handler for the `Initialize` event.
- To create an `Initialize` event handler for the `Screen1` component, go to the `Screen1` drawer in the Blocks column and select the `when Screen1.Initialize do` block.

The `Screen's` `Initialize` Event

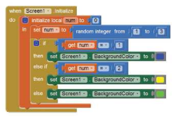
Figure 4-79 shows an example of a `Screen1.Initialize` event handler. It sets the screen's background color randomly to either blue, yellow, or green.

Figure 4-78 The `Screen1` Component's `Initialize` Event Handler (Source: MIT App Inventor 2)



```
when Screen1 BackPressed
do

when Screen1 ErrorOccurred
  component functionName errorNumber message
do

when Screen1 Initialize
do

when Screen1 OtherScreenClosed
  otherScreenName result
do

when Screen1 ScreenOrientationChanged
do
```

Figure 4-79 Example `Screen1.Initialize` Event Handler (Source: MIT App Inventor 2)



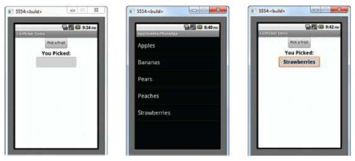
```
when Screen1 .Initialize
do  initialize local num to 0
    in  set num to random integer from 1 to 3
        if get num = 1
        then set Screen1 . BackgroundColor to
        else if get num = 2
        then set Screen1 . BackgroundColor to
        else set Screen1 . BackgroundColor to
```

The `ListPicker` Component

A `ListPicker` component displays a list of items and allows the user to select an item from the list.

Figure 4-80 shows an example app (ListPickerDemo) running in the emulator.

Figure 4-80 The ListPickerDemo App Running in the Emulator (Source: MIT App Inventor 2)



The `ListPicker` Component

`ListPicker` has all the same properties as a `Button` component, plus a couple of extra ones:

•`ElementsFromString`: This property holds the list of items that is displayed when the user clicks the `ListPicker` as seen in Figure 4-81.

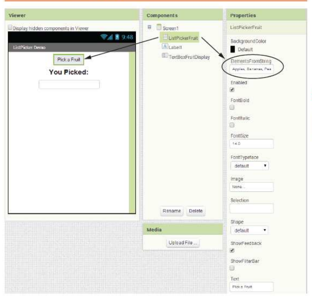
•`Selection`: Once the user selects an item from the list, the selected item is copied into the `Selection` property.

•When the user selects an item from a `ListPicker’s` list, an `AfterPicking` event is triggered.

•Figure 4-82 shows the `AfterPicking` event handler for the `ListPicker`.

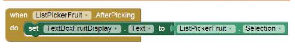
The `ListPicker` Component

Figure 4-81 The ListPicker Component's ElementsFromString Property (*Source:* MIT App Inventor 2)



BackgroundColor: Default
ElementsFromString: Apples, Bananas, Pea
Enabled: ✓
FontBold:
FontItalic:
FontSize: 14.0
FontTypeface: default
Image: None...
Selection:
Shape: default
ShowFeedback: ✓
ShowFilterBar:
Text: Pick a Fruit

Figure 4-82 The `AfterPicking` Event Handler (*Source:* MIT App Inventor 2)



```
when ListPickerFruit.AfterPicking
do  set TextBoxFruitDisplay.Text to ListPickerFruit.Selection
```

The `CheckBox` Component

- The `CheckBox` component appears as a small box with some accompanying text.
- In the *Designer*, `CheckBox` components are found in the *User Interface* section of the Pallet.
- You will mostly be concerned with the `Text` and `Checked` properties.
- The `Text` property determines the text that is displayed next to the small box.
- The `Check` property indicates whether the component is checked or unchecked.

The `CheckBox` Component

Figure 4-89 shows the PizzaToppings app in the Designer and Figure 4-90 shows how it initially appears in the emulator.

Figure 4-89 The PizzaToppings App in the Designer (*Source:* MIT App Inventor 2)

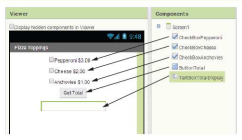
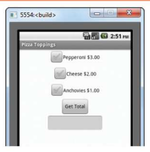


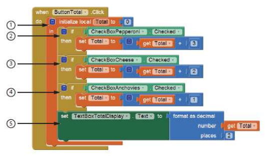
Figure 4-90 The PizzaToppings App Initially in the Emulator (*Source:* MIT App Inventor 2)



The CheckBox Component

Figure 4-91 shows the apps workspace in the Blocks Editor.

Figure 4-91 The App's Workspace in the Blocks Editor (Source: MIT App Inventor 2)



```
when ButtonTotal .Click
do  initialize local Total to 0
    in  if CheckBoxPepperoni . Checked
        then set Total to  get Total + 3
        if CheckBoxCheese . Checked
        then set Total to  get Total + 2
        if CheckBoxAnchovies . Checked
        then set Total to  get Total + 1
        set TextBoxTotalDisplay . Text to  format as decimal
                                             number get Total
                                             places 2
```

The CheckBox Component

1. This block initializes a local variable named `Total`, with the initial value of 0.
2. This `if then` block determines whether the `CheckBoxPepperoni` component is checked. If so, 3.00 is added to the `Total` variable.
3. This `if then` block determines whether the `CheckBoxCheese` component is checked. If so, 2.00 is added to the `Total` variable.

The CheckBox Component

4. This `If then` block determines whether the `CheckBoxAnchovies` component is checked. If so 1.00 is added to the `Total` variable.
5. This block displays the value of the `Total` variable, rounded to two decimal places, in the `TextBoxTotalDisplay` component.

The CheckBox Component

The Changed Event

•Any time a `CheckBox` component's `Checked` property changes, a `Changed` event happens.

•In the Blocks column, click the name of the `CheckBox` component then select the block for the `Changed` event handler.

The CheckBox Component

The Changed Event

- Figure 4-93 shows a PizzaToppings2 app in the Designer and Figure 4-94 shows how it initially appears in the emulator.
- This app serves the same purpose as the Pizza Toppings app except it does not require the user to click a button to calculate the total cost.

The CheckBox Component

Figure 4-93 The PizzaToppings2 App in the Designer (*Source:* MIT App Inventor 2)

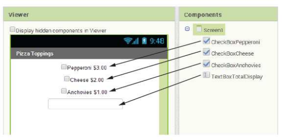
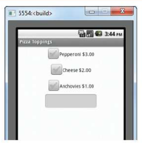


Figure 4-94 The PizzaToppings App Initially in the Emulator (*Source:* MIT App Inventor 2)

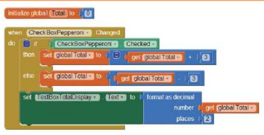


The CheckBox Component

The Changed Event

- This app creates a global variable named `Total`, initialized to 0.
- The app also has a Changed event handler for each of the `CheckBox` components.

Figure 4-95 The `CheckBoxPepperoni.Changed` Event Handler (*Source:* MIT App Inventor 2)



```
initialize global Total to 0

when CheckBoxPepperoni.Changed
do  if CheckBoxPepperoni.Checked
    then set global Total to get global Total + 3
    else set global Total to get global Total - 3
    set TextBoxTotalDisplay.Text to format as decimal
                                        number get global Total
                                        places 2
```

The CheckBox Component

The Changed Event

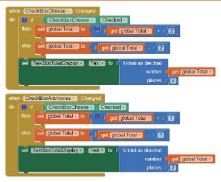
- The event handler works like this:
- If `CheckBoxPepperoni` is checked, then add 3.00 to the `Total` variable.
- Otherwise, subtract 3.00 from the `Total` variable.
- Display the `Total` variable rounded two decimal places.

The `CheckBox` Component

The Changed Event

Figure 4-96 shows the `CheckBoxCheese.Changed` and `CheckBoxAnchovies.Changed` event handlers, which worked in a similar fashion.

Figure 4-96 The `CheckBoxCheese.Changed` and `CheckBoxAnchovies.Changed` Event Handler (*Source:* MIT App Inventor 2)



```
when CheckBoxCheese .Changed
do  if CheckBoxCheese . Checked
    then set global Total to get global Total + 2
    else set global Total to get global Total - 2
    set TextBoxTotalDisplay . Text to format as decimal
                                      number get global Total
                                      places 2

when CheckBoxAnchovies .Changed
do  if CheckBoxCheese . Checked
    then set global Total to get global Total + 1
    else set global Total to get global Total - 1
    set TextBoxTotalDisplay . Text to format as decimal
                                      number get global Total
                                      places 2
```