PLTW COMPUTER SCIENCE

Activity 1.1.3

Conditionals and Event-driven Programming: Happy Balance

goals

- Learn how to use conditionals to make choices in a program
- Learn to use modifiers to create chained conditionals
- Personalize the user interface and features of an app incrementally
- Apply coding fundamentals to create algorithms
- Develop an app independently for creative expression

description of app

Modify the app you created in the previous activity to include a random number generator and a countdown timer so that the user can play without needing a second player. You may also choose to add other modifiers that personalize the app and make the game more challenging.

Essential Questions

1. Why do you think all decisions in programs are narrowed down to two options, yes or no, true or false?
2. What information is being hidden or abstracted by the program?
3. How did you deal with challenges you were confronted with?

Essential Concepts

- Conditionals and Chained Conditionals
- Algorithms, Variables, Arguments, Procedures, Operators, Data Types, Logic, and Strings

Resources

Activity 1.1.3 Student Files

1.1.3 Challenge: Additional App Features

Conditionals

20 Questions

Conditional statementsare how programs make decisions. Conditional statements are questions that the programmer writes to determine what the app does next. They rely on Boolean expressionsthat force the answer to be one of two values, such as true or false. Based on the result of the Boolean expression, the program determines what to do next.

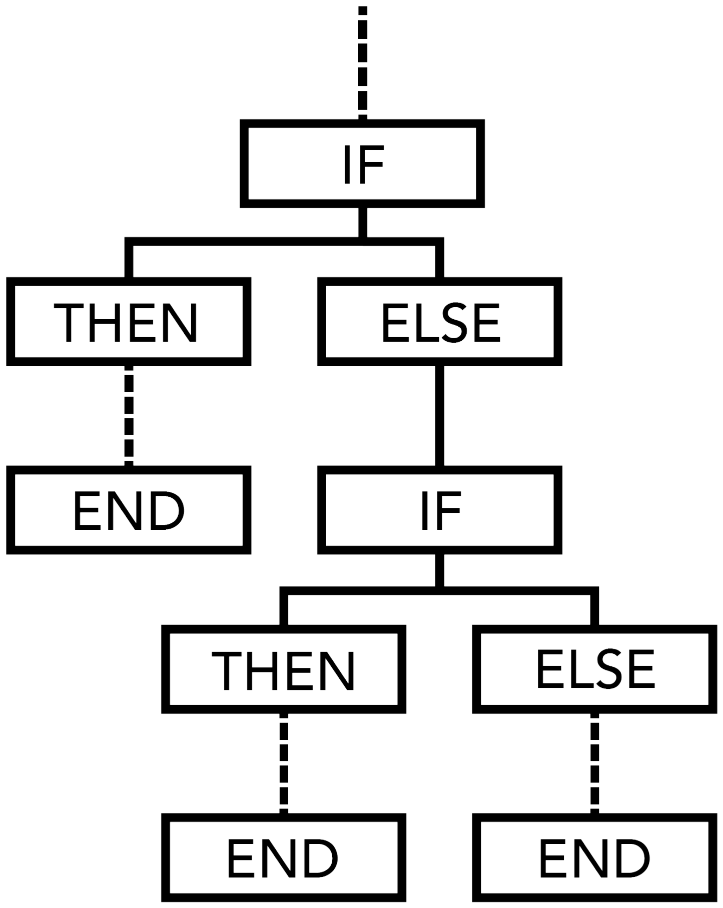
20 Questions is a classic game played since the nineteenth century. To play 20 Questions, one person thinks of an object and the other players can ask a maximum of 20 questions to help them guess what the person is thinking of.

You can actually get very sophisticated results or information with just a few well-designed yes/no or true/false questions.

1. With an elbow partner, play 20 Questions to see if you can determine what object in the classroom the other person is thinking of.
2. Working with the same elbow partner, write out the questions asked in your 20 Question game to show how you narrowed the list of potential objects down to a single object.
3. Talk with others in the class to see the questions they used.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

If-Then-Else Conditional Statements



An essential idea in computer science is the *if-then-else* conditional construct. In the conditional construct:

- A question is asked
- A Boolean conditional makes a comparison and decides whether it is “True” or “False”
- If “True”, the logic flows to the “Then” branch
- If “False”, the logic flows to the “Else” branch

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

In this activity, you will use conditionals and Boolean-type data at a lower level of abstraction than before, to have the program make decisions in your game, based on the user inputs.

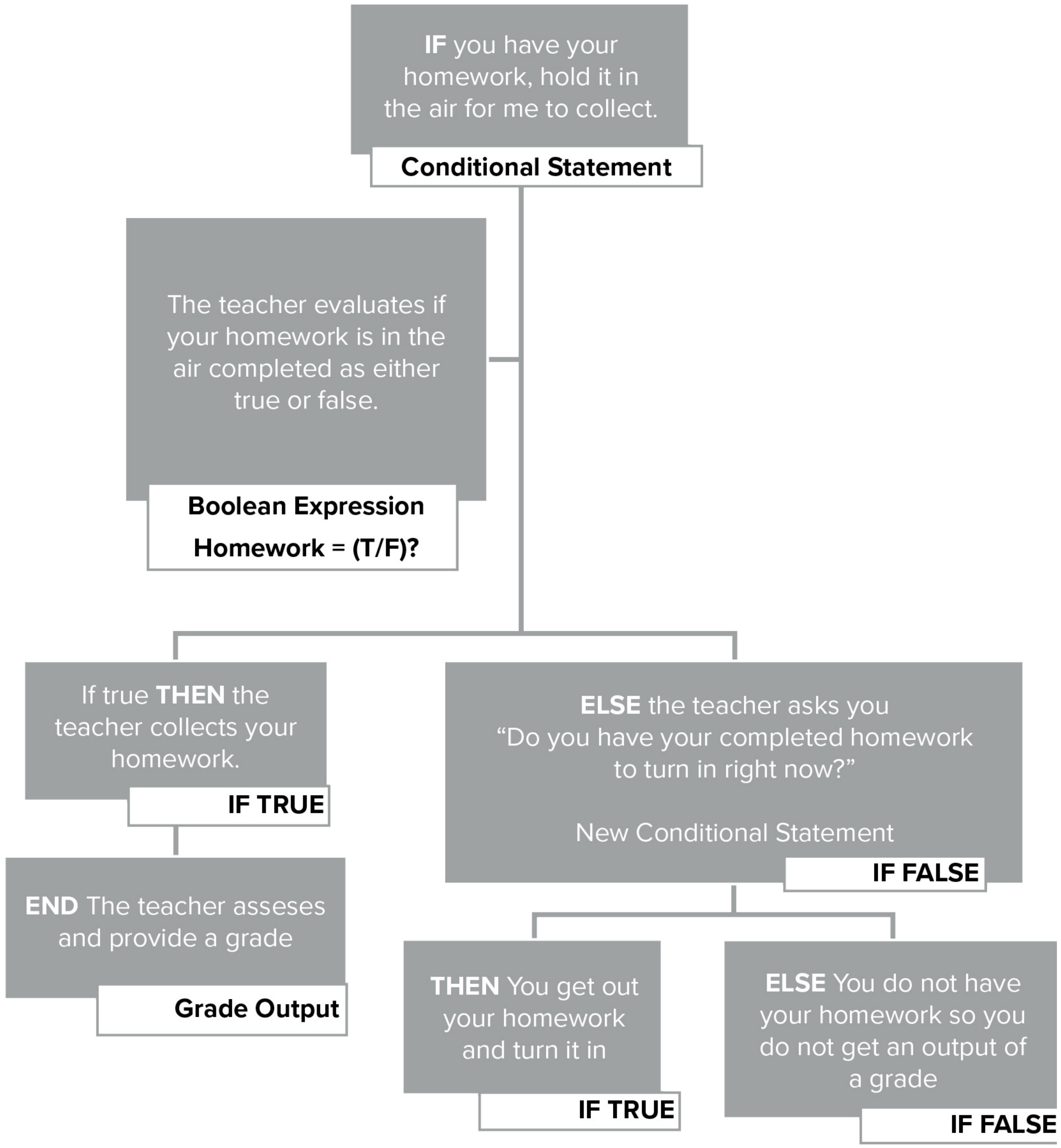
Example: Conditionals for Turning in Homework

Conditional statement: **If** you have your homework, **then** please hold it in the air for me to collect. **Else**, sit quietly until the homework is collected.

- Boolean expression of input: Does the student have the homework?
- **If** TRUE, your completed homework is in the air, **then** the teacher collects your homework. (The output would eventually be your grade.)
- **If** FALSE, your completed homework is not in the air, **then** the teacher does not collect your homework to grade. (There is no input to get an output back from.)

- **Else**, you do not have your homework, so you should be sitting quietly.

Computer scientists sometimes use flowcharts to help map out how to program these situations. Here is an example of what a flowchart might look like for the conditional statements above:


IF you have your homework, hold it in the air for me to collect.
Conditional Statement
The teacher evaluates if your homework is in the air completed as either true or false.
Boolean Expression
Homework = (T/F)?
If true THEN the teacher collects your homework.
IF TRUE
END The teacher asseses and provide a grade
Grade Output
ELSE the teacher asks you "Do you have your completed homework to turn in right now?"
New Conditional Statement
IF FALSE
THEN You get out your homework and turn it in
IF TRUE
ELSE You do not have your homework so you do not get an output of a grade
IF FALSE

Iterating an App

Design Overview: Happy Balance Game

User Story

Add to the *HappyAccelerometer* app that you created in Activity 1.1.2, and turn it into the Happy Balance Game - a game that tests the user's ability to balance their device on their foot.

App Overview

The point of the game is to keep the happy face in the center of the screen by balancing the device on your foot. Challenge the user to balance their mobile device by programming a sound to play whenever the happy face hits an edge of the screen. You will program a different sound to play when the face hits each different edge, so the user knows which edge they hit. You will also make an end game event that stops the motion and displays a sad face when the *HappyImage* hits an edge. A user can restart the game after tapping on the sad face to get a prompt to play again.

Design Update

Add the following features to transform the Happy Accelerometer into the Happy Balance Game.

- ☐ When the *HappyImage* hits the edge of the screen, a sound plays to indicate the hit edge to the user.
- ☐ The *HappyImage* is no longer visible when it hits the edge.
- ☐ A *SadImage* becomes visible when the *HappyImage* hits the edge.
- ☐ The *ReplayLabel* becomes visible to give directions on how to replay when the *HappyImage* hits an edge.
- ☐ Clicking on the *SadImage* makes the notifier pop up to prompt the user to

play again.

- ☐ Clicking the **No** option on the notifier means nothing happens.
- ☐ Clicking the **Yes** option resets the game to play again.

Initial Backlog Breakdown
The user needs to be able to:

- ☐ Hear a sound specific for each edge of the tablet
- ☐ Have the game end when an image hits the edge of the screen
- ☐ Have the option to restart the game to play again
-

User Testing and Feedback
You will be sharing what you develop with others in your class to get feedback to improve your app. Later in the unit and course, you will continue to capture user stories to learn about people's app preferences, ways to make your developments more interesting, unique, and challenging.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

4. Navigate to MIT App Inventor and log in.
5. Open the *HappyAccelerometer* app.
6. Select **Project** > **Save project as**.
7. Save the app name based on your teacher's directions.
8. Start the download of the activity's media source files:
9. Make the following updates to the user interface in the *Designer* view.
   - Change the app name to *BalanceChallenge* under the *Screen1 Properties*.
   - Upload the sound files to the *Media Palette* in the *Designer* view.
10. Add the four *Sound* components from the *Media* drawer. *Sound* components are non-visible components that allow sounds to play.
11. Upload the sound files provided in the source files. Rename the *sound* components to match the *sound* that you uploaded to that component:
    - left
    - right
    - top
    - bottom

12. Download the *Sad.png* image file.

13. Add an *ImageSprite* from the *Drawing and Animation* drawer:

    - Upload the *Sad.png* from your source files.
    - Set the *Properties* for the *Sad.png* image uploaded from the activity's source files.
    - Set the Width and *Height* to "50" pixels each.
    - Rename the component "SadImage".

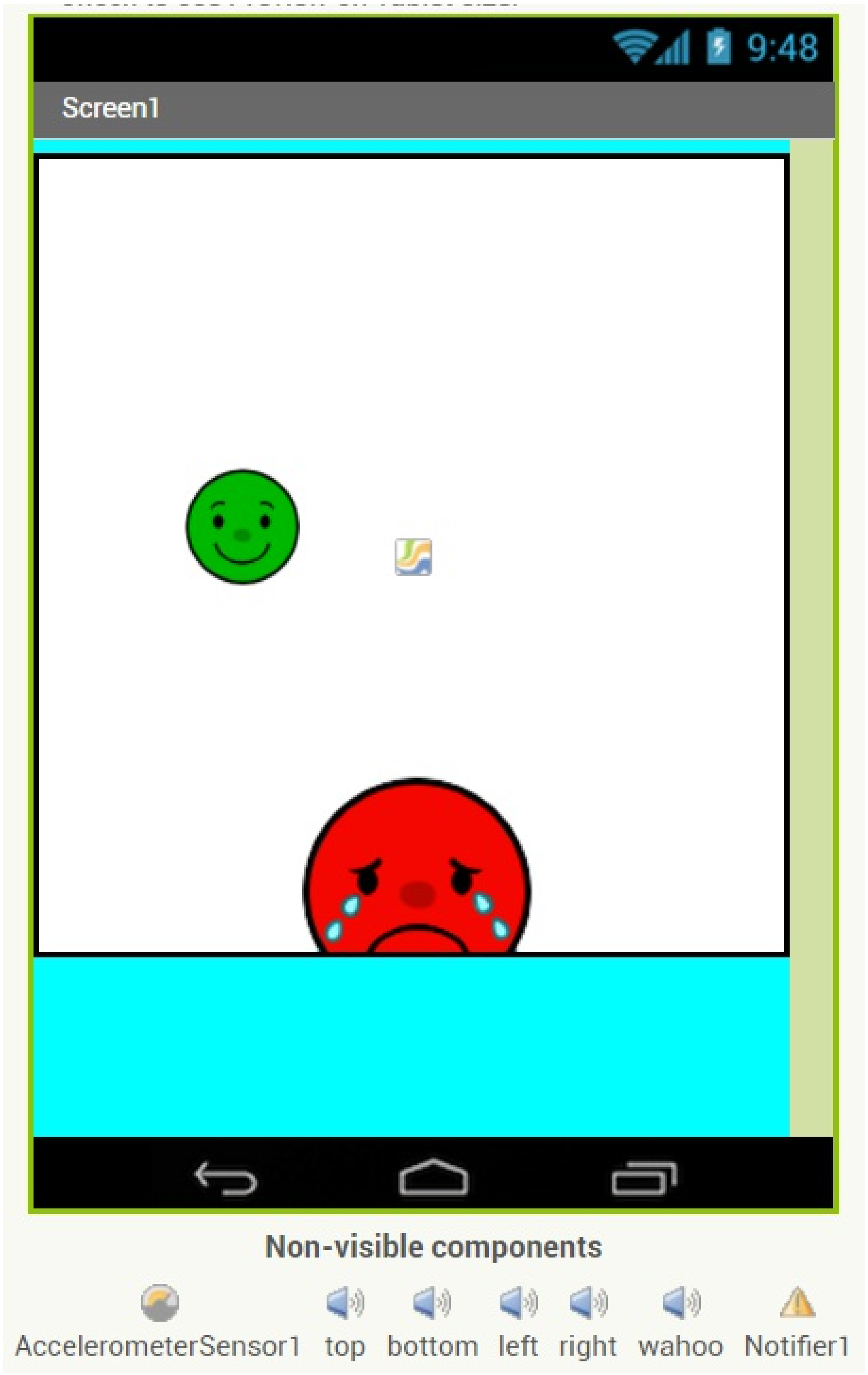
14. Adjust the *Properties* of the *HappyImage* and *SadImage* in the *Designer* view to make sure that neither the happy face nor the sad face is touching the edge of the tablet.

15. Add a *Notifier*. A *Notifier* is a non-visible component from the *User Interface* drawer that creates a screen popup. You will use a *Notifier* to allow the user to touch the *SadImage* sprite and load a new game.

16. Add a *Label,* referred to as *ReplayLabel*:

    - In *Properties* for the *ReplayLabel*, uncheck the *Visible* box.
    - Set the *text* to "Click the sad face to play again".

17. Set up the design of your user interface that you think will work best. Here is an example of what the user interface may look like; just make sure you have all the needed components in the *Designer view*.



Boolean Statements

Screen Initialization Setup

Now that the design is set up, go to the *Blocks* view to start programming. The first thing to address is making the *HappyImage* visible and making sure the *SadImage* is not visible when the game first starts. This is done by using Boolean statements to set the *Visible* value to "true" or "false".

True and false conditional statement blocks are in the *Logic* drawer; the statement gives the computer a Boolean statement of logic about how the program should behave.

You will find four other types of Boolean logic in the *Logic* drawer:

not
equal to
and
or

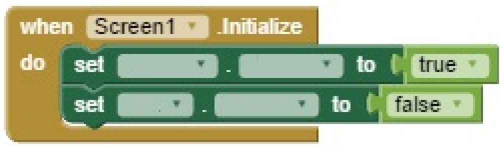
These allow you to check multiple conditions at the same time. Such as, if it is after 5 p.m. and people are hungry, then eat dinner. You will learn more about how to use these operators as you compare values later in the activity.

18. Click the **Screen1** component in the *Blocks* Palette. This gives you access to the prebuilt event handlers and procedures specific to this component.

19. Drag the *Screen1.Initialize* event handler into your program.

20. Inside the *Screen1.Initialize*, there are a few events to set up:

    - Set the *HappyImage.Visible* property to “true”.
    - Set the *SadImage.Visible* property to “false”.



    Important: It is possible to right-click and duplicate existing blocks, then just change the name inside the drop-down list of the logic block.

21. Test and debug your app. All the previous app features should still be working, and the happy face should be visible, but not the sad face.

    - ☐ The app shows the accelerometer numbers.
    - ☐ The app shows the X and Y coordinates.
    - ☐ The *HappyImage* moves across the screen.
    - ☐ The happy face is visible, but not the sad face.

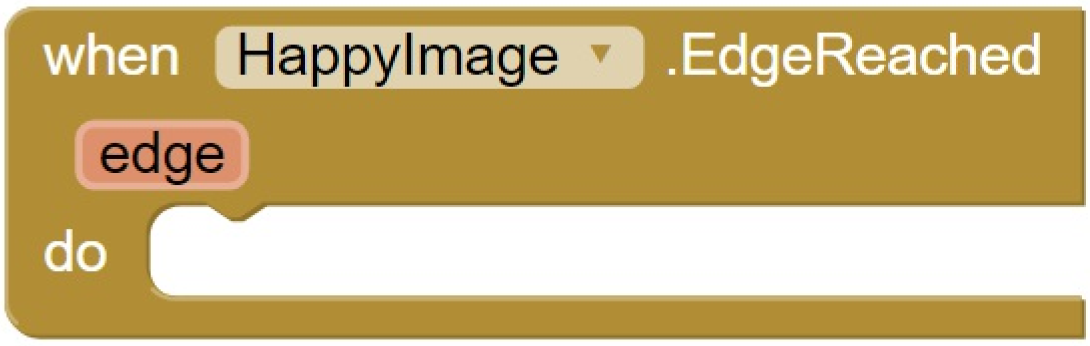
Programming Iteration

Because you will be modifying your app, it is important to do a **Save As** with a new name. In this way, you will be able to try out new features without losing the progress of having a working app. These iteration saves are similar to creating your own checkpoints to return to if you need to review or reuse old code.

It is important to begin thinking of the users’ needs. If users are balancing a device on their feet, will they always be able to see when they lose? To make the game more usable, add sounds to notify users when they have lost their balance and the happy face hits the edge of the screen. These sounds will also provide a different sound for each edge to let users know which edge of the screen the face hit.

Programming HappyImage Sprite Movement and Sounds

22. Click the **HappyImage** component to access the *HappyImage.EdgeReached* event handler.



23. Call the *bounce* procedure of the *HappyImage* to get the edge values.

    - The *bounce procedure* is a prebuilt procedure that tells the sprite to switch its direction based on the angle at which it hits the screen edge (so it looks like it is bouncing off a wall the way it would in the physical world).
    - The *edge* variable is a prebuilt variable that contains the (X,Y) values (arguments) that describe the edge of the *Canvas*. As a reminder: To find the needed procedure and variable, focus on the component that is experiencing the event to know what drawer will have the blocks.

Chained Conditionals and Operators

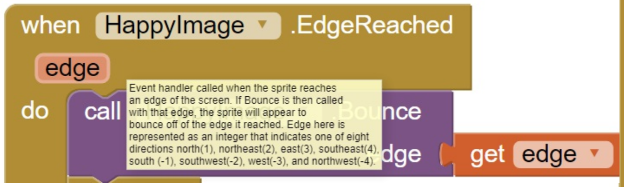
Since you have four edges the sprite could hit, you have four conditionals to check as the program is running. Because all the events are similar, you might want to check them all in the same place each time the program runs through a single event handler.

To get a different sound to play when the *HappyImage* hits different edges, you will use a chained conditional statement. A chained conditional statement is a series of conditionals that a program evaluates in sequence until stopping on a condition that is true.

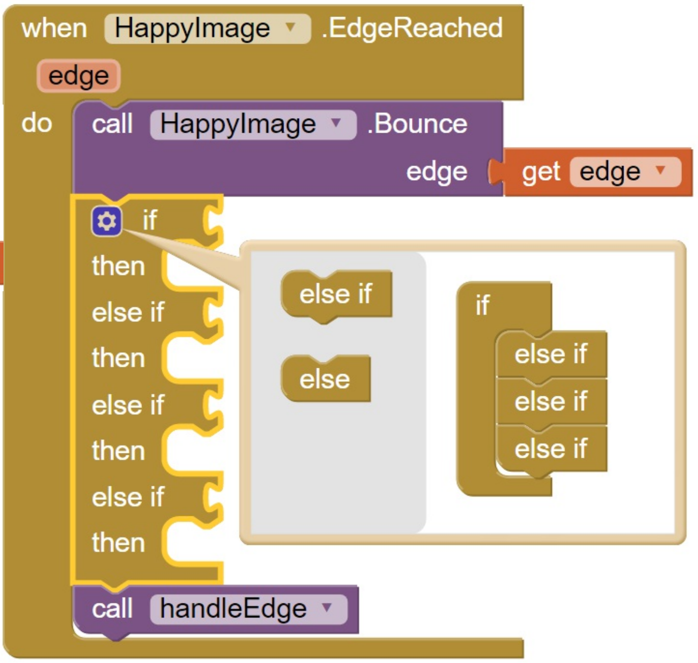
In this case, the chained conditional will evaluate which edge the *HappyImage* hit to know which sound to play.

24. Inside the *Control* drawer, find the *If then* block and add it to the *EdgeReached* event handler.
25. Use the *mutator* box to add enough *else if* block statements to handle all four of the edges.
26. Hover your pointer over the event handler to

open help text that tells you the value of each edge. While App Inventor gives you eight options total, you will only use four.



Event handler called when the sprite reaches an edge of the screen. If Bounce is then called with that edge, the sprite will appear to bounce off of the edge it reached. Edge here is represented as an integer that indicates one of eight directions north(1), northeast(2), east(3), southeast(4), south (-1), southwest(-2), west(-3), and northwest(-4).



Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Operators

For the chained conditional to know which part to act on next, you need to add logic operators. Your goal is to assign the left sound to the *HappyImage* hitting the left edge and so on for the other sounds and edges.

The happy face is always moving, so the values describing the location on the happy face are always changing. This is why you call the variable, *edge*. When you call the variable, the program returns the argument (current value) of the variable.

You will want to compare the current location to the integer that defines the edges for each of the four edges in your if statement.

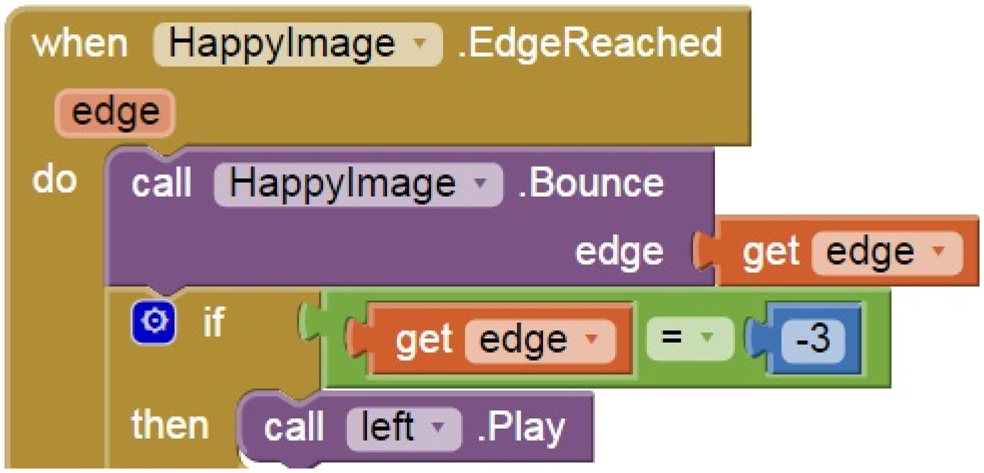
Important: If the variable edge is equal to the integer that represents the left edge (–3), then call the play procedure and left.

27. Add the Boolean expression from the Logic drawer that will evaluate whether two things are equal.
28. Hover over the edge variable in the *HappyImage.EdgeReached* event handler.
29. Grab and drag out the *get edge* block, which will return the argument (current value) that you will compare to the integer that represents the edge.
30. Add a blank *Math* block where you can enter the integer that represents the edge.

    Important: If you are not sure what the value is, hover over the *HappyImage.EdgeReached* event handler for information about the values of that event handler.

31. Set up the first *if* statement to have the get edge argument in a logical operator be compared to the value of that edge.

32. In the *then* section, under the *if* block you just set up, call the appropriate sound procedure to play.



```
when HappyImage .EdgeReached
  edge
do  call HappyImage .Bounce
                   edge  get edge
    if   get edge = -3
    then call left .Play
```

33. Based on how you set up the left side sound to play, write out the rest of the chained conditional. Share it with your elbow partner.

34. Once you agree on what to do, program the rest of your chained conditional to have the appropriate sound play when *HappyImage* hits all the edges.

35. Test, debug, and adjust.

    To be able to keep testing, you may need to use the Reset options under the Connect menu at the top of the App Inventor window. If your code updates do not seem to be doing anything, it is good debugging practice to reset the connection to your code.

    - ☐ The app should show the accelerometer numbers.
    - ☐ The happy face is visible, but not the sad face.
    - ☐ The app should show the X and Y coordinates.
    - ☐ The *HappyImage* moves across the screen.
    - ☐ A sound identifies what edge was hit by name.

Important: To establish a history of successful program iterations, remember to save your project with a new name before you continue.

End Game Procedure

A game needs a start and an end. You are going to add an event that will end the balance game. The end-of-game event is when the *HappyImage* hits any edge. The following things should happen at the end of the game:

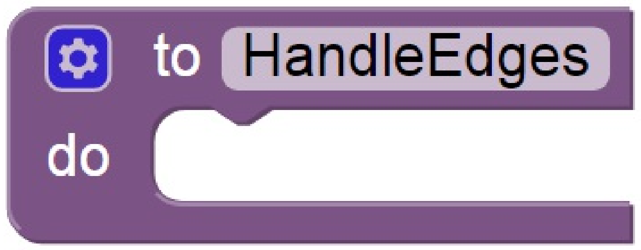
- ☐ All movement stops. (Disable the accelerometer.)
- ☐ The sad face appears.
- ☐ The happy face disappears and then reappears in the center of the screen.
- ☐ A message appears telling the user how they can choose to play again.

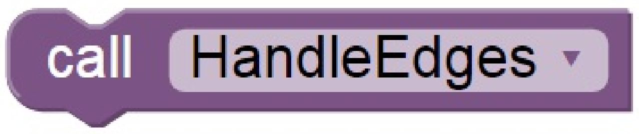
Until now, you have been using many prebuilt procedures in App Inventor. You will now create your own procedureto end the game when the *HappyImage* hits any edge.

You need a procedure that will handle all four edges of the tablet the same when the *HappyImage* hits an edge. The advantage of a procedure is that multiple places in the code can call the procedure to run.

There are two types of procedures in the *Procedures* drawer: *to do* and *to result*.

36. Click the *Procedures* block drawer and drag out the *to procedure do* block.
37. In place of *procedure*, type the name “HandleEdges”. You have now created your own procedure (not a prebuilt one).
38. Go back and look in the *Procedures* block drawer. You will now see a *call* procedure block named “HandleEdges”.





Important: The call procedure is only available after you have made the procedure. You can drag the call procedure block around the code and add it as many times as you want (anywhere you want the same instructions to execute).

Using Conditional Logic (True or False) in Your Procedure

39. Inside the *HandleEdge* procedure, you need to establish the default value for item visibility. Consider what you want to be visible and when. Add the Boolean values of True or False to the appropriate components.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Arithmetic Operators in the Procedure

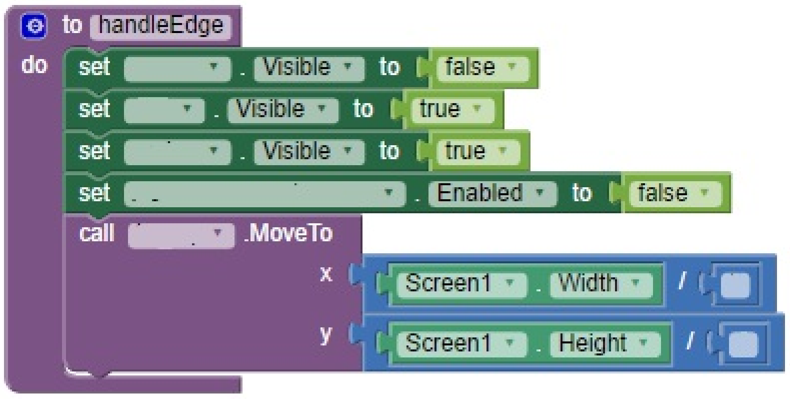
If a rectangle is 10 inches in height and 4 inches in width, you could calculate the halfway point on its axes as 5 inches for the Y value and 2 inches for the X value.

What if you did not know the exact values for the height and width, or the values were not round integers but floats? Could you create a mathematical expression to find the halfway point for each?

By using an arithmetic operator in App Inventor, you can get the height and width of the *Screen1* and find the middle in the X and Y directions without having to know the values.

- Half of the *Screen1.Width* would be the middle in the X direction.
- Half of the *Screen1.Height* would be the middle in the Y direction.

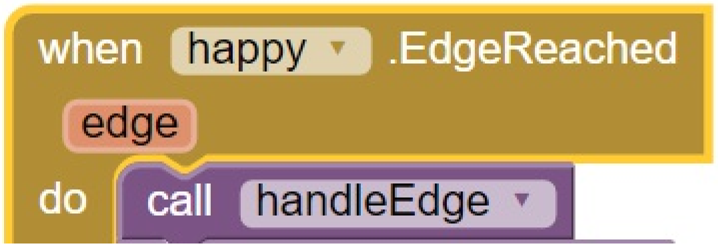
40. To call the *HappyImage.MoveTo* procedure, drag out the call block and add it to your *handleEdge* procedure. You are putting a procedure within a procedure, again using abstraction to focus on only one part at a time!
41. Reset the *HappyImage* sprite to the middle of the screen horizontally:
    - Connect an arithmetic operator from the *Logic* drawer to the *x* socket of the procedure.
    - Add a *Screen1.Width* block inside the arithmetic operator.
    - Add an integer that will divide the screen in two.
42. Reset the *HappyImage* sprite to the middle of the screen vertically:
    - Connect an arithmetic operator from the *Logic* drawer to the *x* socket of the procedure.
    - Add a *Screen1.Height* block inside the arithmetic operator.
    - Add an integer that will divide the screen in two.



43. Test your app. Spend only a minute on testing your app, because you may notice it does not do anything different.

    Important: Your app did nothing different when the *HappyImage* hit the edge, because the procedure is not called in the *HappyImage.EdgeReached* event handler. The procedure is never told to execute.

44. Call the *HandleEdge* procedure in the *HappyImage.EdgeReached* event handler to make the app go through the procedure steps.



45. Test, debug, and adjust.
    - ☐ The app shows the accelerometer numbers.
    - ☐ The app shows the X and Y coordinates.
    - ☐ The *HappyImage* moves across the screen.

- ☐ The happy face is visible, but not the sad face.
- ☐ A sound identifies what edge was hit by name.
- ☐ When the *HappyImage* hits an edge, it is no longer visible.
- ☐ When the *HappyImage* hits an edge, the *SadImage* and the *ReplayLabel* become visible.
- ☐ The accelerometer is disabled, so there is no more motion.
-

If you think that your code updates do not seem be doing anything, it is good debugging practice to reset the connection between MIT App Inventor and the device.

After a reset, if it still not working, go through the debugging strategies as discussed in previous activities.

Important: Remember to save your project with a new name before you continue.

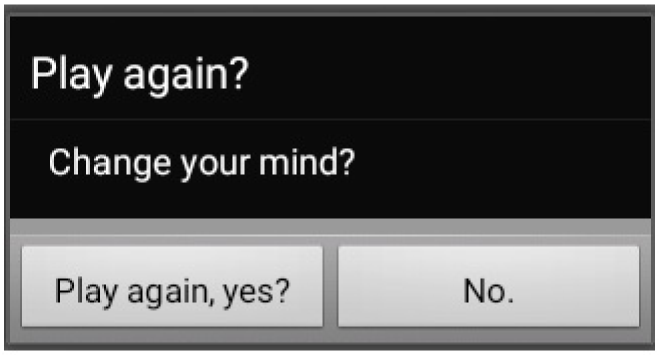
PLTW DEVELOPER'S JOURNAL Outline the steps to create and use a procedure.

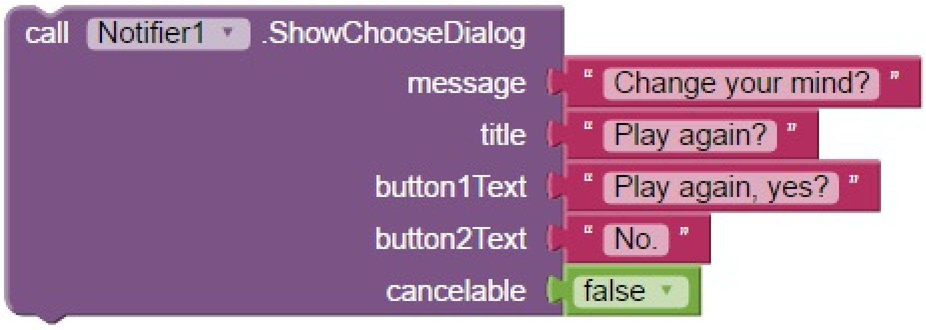
Conditional Statements for Replay

You are now going to use the *Notifier* pop-up to allow the user to touch the *SadImage* and select **Play again** to restart the game.

46. Drag out the when *SadImage.Touched* event handler.
47. From the *Notifier1 drawer*, drag out the call *Notifier1.ShowChooseDialog* procedure block to add to the *SadImage.Touched* event handler.
48. Evaluate the similarities between the notifier procedure and the notifier that users see.

    The message in the strings can vary, but you want a professional way of asking whether the user wants to play again or keep the end game screen.





```
call Notifier1 .ShowChooseDialog
    message      "Change your mind?"
    title        "Play again?"
    button1Text  "Play again, yes?"
    button2Text  "No."
    cancelable   false
```

49. In the *notifier* procedure, use strings to set up the *message*, *text*, *title*, and the two button texts.

50. Set the *cancelable* option to *false* with a logic block.

    Important: You need to add code to make the program behave differently depending on which *notifier* button the user presses.

51. In the *Notifier* drawer, pull out the event handler that handles *.AfterChoosing*. This event handler triggers as soon as the user presses a button in the notifier pop-up.

52. Add a conditional *if*, *then* statement that does the following:

    - IF get choice = "same string you used in the button to play again"
    - THEN:
        - set *Accelerometer* enabled to true
        - set *HappyImage* visible to true
        - set *Accelerometer* enabled to true
        - set *SadImage* visible to false
        - set *ReplayLabel* visible to false

53. Test, debug, and adjust.

Important: To test, you might need to use the **Hard Reset** option under the Connect menu or force your app to quit on the Android™ device.

- ☐ The app shows the accelerometer numbers.
- ☐ The app shows the X and Y coordinates.
- ☐ The *HappyImage* moves across the screen.
- ☐ The happy face is visible, but not the sad face.
- ☐ A sound identifies what edge was hit by name.
- ☐ When the *HappyImage* hits an edge, it is no longer visible.
- ☐ When the *HappyImage* hits an edge, the *SadImage* and the *ReplayLabel* become visible.
- ☐ The accelerometer is disabled, so there is no more motion.
- ☐ When you click on the sad face, the notifier pops up.
- ☐ If you click the no option, nothing happens.
- ☐ If you click the yes option, everything resets and you can play again.
- ☐ The *HappyImage* is visible and the *SadImage* is not visible at the start of the new game.

Important: Remember to save your project with a new name before you continue.

Personalize Your App

It is important to personalize this app so that while you and others in your class play the game, you can identify your app. Make sure that you keep all design elements professional.

54. In *Designer view*, select **Screen1** under the *Components* list, and look at the *Properties* of that component.

55. *BackgroundColor* – Change the background color of your screen to help identify your program.

56. *Icon* – The icon image is what you tap on the tablet screen to start an app and also what is displayed on a device after download.

    - Make sure the icon is identifiable as the particular app and school appropriate.
    - If directed by your teacher, you may use the Happpy Balance app icon picture located in the **source files**.
    - First, upload any image file you want to use. Then, select the icon box to select the name of the image you uploaded.

57. When your app is working as outlined in this activity, you can share, back up, and/or download the app as instructed by your teacher.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

58. As directed by your teacher, try out your balance!

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Challenge

Challenge yourself by choosing some **additional app features**.

Conclusion

1. Explain why procedures are considered an abstraction in your program.
2. Why do you need to set the values at the beginning of a procedure?
3. How did you interpret and respond to the **essential questions**? Capture your thoughts for future conversations.

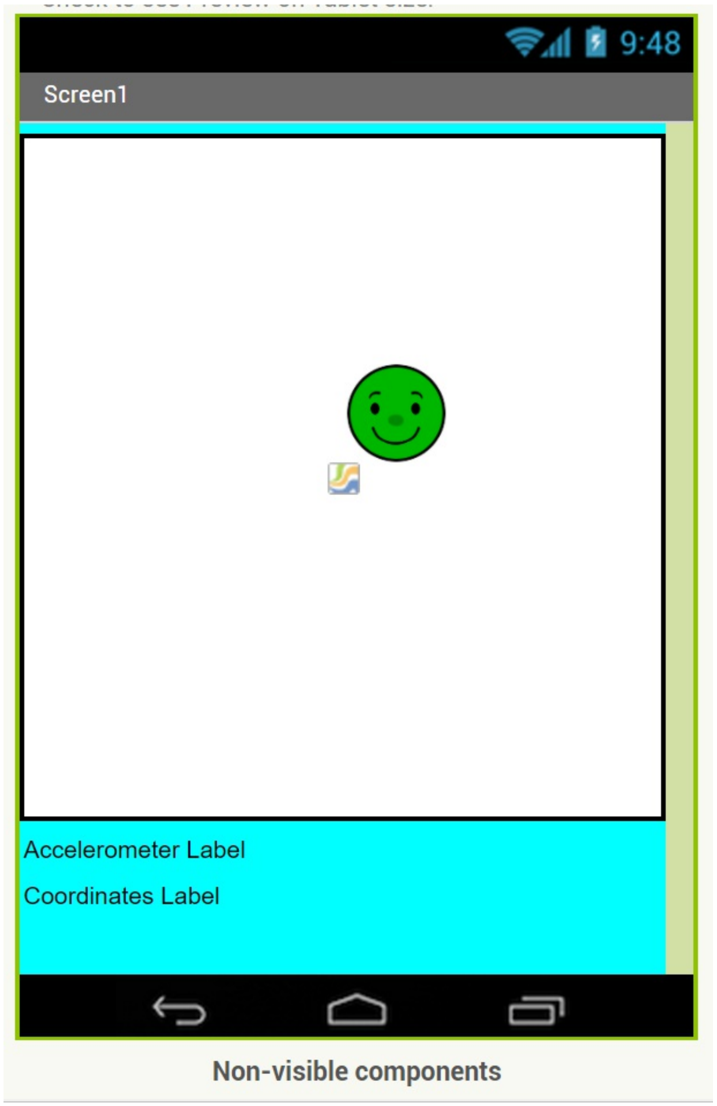
Challenge: Additional App Features

Resources

1.1.3 Challenge Student Files

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

To 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 – Universal Movement Math

Goal

Make the app work regardless of the type of device it is played on.

Design Needs

- Modify existing blocks.

Do

- If you did not already done so, use *Math* blocks to make the *HappyImage* reset back to the center of the screen when someone chooses to play again.
- If this game is played on different devices, the programmer would not know the exact x and y locations and dimensions of their screens. (Tablet screens are larger than phone screens). It is important to use arithmetic operators which give your program flexibility and not always rely on hard coding the values.
- Once it is working, save your project, adding an “A” to the end of the name to show you completed this challenge.

Challenge B – Coordinate Color

Goal

Add buttons to change the color of the pen.

Design Needs

- Modify existing blocks.

Do

- Use conditional statements to add some difficulty, such as having the *HappyImage* move faster or slower, or having the background colors change depending on the location of the *HappyImage* on the screen.
- Use coordinates or math, but keep in mind that coordinates might limit the playability to only your device.
- Once the app is working, save your project, adding a “B” to the end of the name to show you completed this challenge.

Challenge C – Sound Warning

Goal

Have the app play a warning sound when the sprite is approaching the edge, such as when you would have the colors change in Challenge B.

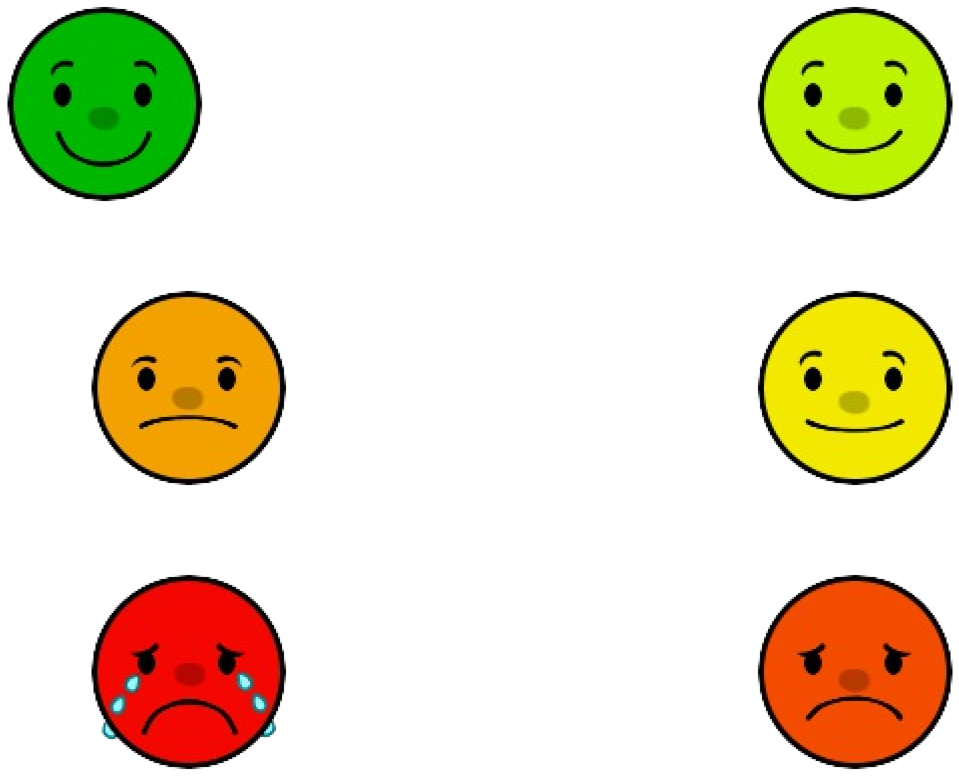
Design Needs

- Modify existing blocks.
- Add one additional sound block that can change the sound it plays.

Do

- Use conditional statements to play warning sounds when an edge is almost reached, so the player has a warning of which edge they are about to hit.
- Use coordinates or math, but keep in mind that coordinates might limit the playability to only your device.
- Once the app is working, save your project, adding a “C” to the end of the name to show you completed this challenge.

Challenge D – Face Progression



Goal

Have the app play a warning sound when it is approaching the edge, before it hits the edge, such as when you would have the colors change in Challenge B.

Design Needs

- Modify existing blocks and add additional face images.

Do

- Download the additional face images from the source files:
  - *face02.png*
  - *face03.png*
  - *face04.png*
  - *face05.png*

Having the image lock up and transform into a frowny face is a pretty harsh end condition to the game.

- Download the extra faces for this activity so you can animate a transition between happy and sad. This way, you can provide the user a short window in which to correct their balance before the sprite reaches the edge of the screen.
- Instead of adding new image sprites, use conditionals to change the face image displayed in a single image sprite.
- Once the app is working, save your project, adding a “D” to the end of the name to show you completed this challenge.

Challenge E – Sound Countdown

Goal

At regular intervals, have a sound play so the user knows how much time has passed while they play.

Design Needs

- Add sound.

Do

- Download the sound files:
  - *SayWhoohoo.mp3*
  - *SayYay.mp3*
- Use the *clock* component and intervals to have a sound play after a set amount of time.
- Once the app is working, save your project, adding an “E” to the end of the name to show you completed this challenge.

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.