Food Chase Game

1. Synopsis

Students will create a game app where the user controls a red ball with a flinging action. The ball chases food ImageSprites to "eat" and grow in size. A green ball must be avoided. If the red ball collides with the green ball, the game ends. Students will get more practice with procedures, creating a Restart procedure to reset ImageSprites. They also will learn how to use conditionals to check for collision of ImageSprites with the red ball. They will also use a conditional to test for user input when the game ends.

2. Learning Objectives

After completing this unit, students will be able to:

1. Code a game app that includes animated sprites.
2. Use conditionals to correctly check two values within a program.
3. Demonstrate abstraction with a procedure.
4. Use variables correctly to store and retrieve data.
5. Improve their computational identity by making an app that can be shared with friends and family.
6. Work collaboratively with a partner to create a mobile app.

3. Mapping with the CSTA Standards

This table show the alignment of this unit with the intended learning outcomes of the CSTA CS Standards:

2-AP-10	Use flowcharts and/or pseudocode to address complex problems as algorithms. [C] AP: Algorithms [P] Abstraction (4.4, 4.1)	Flowchart used to break down RedBall.CollidedWith event.
2-AP-11	Create clearly named variables that represent different data types and perform operations on their values. [C] AP: Variables [P] Creating (5.1, 5.2)	highScore variable used to store current high Score.
2-AP-12	Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals. [C] AP: Control [P] Creating (5.1, 5.2)	Nested loop is used in RedBallCollidedWith event.
2-AP-13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs. [C] AP: Modularity [P] Computational Problems (3.2)	Student iteratively and incrementally code the parts of the app.
2-AP-14	Create procedures with parameters to organize code and make it easier to reuse. [C] AP: Modularity [P] Abstraction (4.1, 4.3)	Restart procedure is used.

2-AP-17	Incorporate existing code, media, and libraries into original programs, and give attribution. [C] AP: Program Development [P] Abstraction (4.2), Creating (5.2), Communicating (7.3)	Template includes sprite images for student use.
2-AP-18	Systematically test and refine programs using a range of test cases. [C] AP: Program Development [P] Testing (6.1)	Students test and debug each new feature added to app.

4. Learning Prerequisites

Students should have experience with App Inventor from previous units of this curriculum.

5. Lesson Plan (45 minutes x 5)

This unit consists of five 45 minute lessons.

Lesson 1

Time	Activity
10 min	**Introduction to Unit** 1. Ask students how the ball was controlled in the Find the Gold app. Are there ways other than tilting the mobile device to interact with a game app? 2. Explain that they will make a game app called "FoodChase" in this

	unit. It works as follows:
	a. User controls red ball by flinging action
	b. Red ball "eats" food to grow bigger
	c. Red ball avoids green ball, which moves randomly on screen
	d. If green and red balls collide, game is over
	3. Demonstrate the finished app.
	4. Explain that this unit will be a Pair Programming unit, so they will be assigned a partner to work with. Remind the students of the driver/navigator model.
10 min	**Review of Canvas, Ball, and ImageSprite Components in App Inventor** 1. Canvas is your background for sprites to appears and move. 2. Balls/ImageSprites are the elements on the canvas that can be controlled by user interaction and by coding. 3. This app will use the Flung event for the Canvas and transfer the speed and heading of the fling to set the Ball's movement. 4. The app also detects collision between Ball and ImageSprites to change outcomes of the game.
20 min	**Coding of App** Students work with their partners to code Part 1 of app using *Student Guide: Part 1*.
5 min	**Wrap-up** Check in with student groups to see if they are making progress on the app. Ask if anyone is having problems. Explain that students will continue coding in the next lesson.

Lesson 2

Time	Activity
10 min	**Conditionals** Review conditional statements in coding. Students will use **if-then-else**, so both variations should be covered.
5 min	**Any Component**

	Introduce the **Any Component** and **Any ImageSprite** blocks, that allow students to generalize the random placement of **ImageSprites** in the **RedBall.CollidedWith** event.
25 min	**Coding of App** Student groups continue working on the app. Depending on how far they got in Lesson 1, they may either continue with *Student Guide: Part 1*, or start *Student Guide: Part 2*.
5 min	**Wrap-up** Review conditionals and procedures. Check how far students have gotten with their apps.

Lesson 3

Time	Activity
10 min	**Procedures** Students will add a **Restart** procedure in Part 2 of the tutorial. Review with students what a procedure is, how it is useful, and why we use it in this project.
5 min	**Review Conditionals** Review on student progress so far. Review **if** conditionals and **if-then-else** conditionals.
25 min	**Coding of App** Student groups continue working on their apps. Depending on how far they got in Lesson 2, they may either continue with *Student Guide: Part* 2, or do *Student Guide: Part 3*.
5 min	**Wrap-up** Check how far students have gotten with their apps. Review any issues or questions.

Lesson 4

| Time	Activity
10 min	**Introduction to Variables** Introduce variables as a means to store information in a program. Students will add a Label for High Score in the game, and also add a variable to store that information.
10 min	**Adding Variables and TinyDB** 1. Walk students through adding a variable for High Score. 2. Explain to students what TinyDB is and how it is used. 3. Demonstrate that high score can be tracked in a single run of the app, but if they stop the game, and restart it, the value of High Score disappears. 4. Introduce TinyDB, and persistent data. High score can be saved persistently on the mobile device by storing the high score in TinyDB.
20 min	**Coding TinyDB** 1. Students complete flowchart on page 1 of *Student Guide: Lesson 4*. 2. Student groups complete *Student Guide: Lesson 4* to add variables and TinyDB to implement a high score for the game.
5 min	**Wrap-up** Review variables and TinyDB.

Lesson 5

Time	Activity
5 min	**Introduction to Lesson** Student groups may continue to work on their apps, if they still need time. For those who have completed Parts 1-4, they can try some of the challenges.
25 min	**Coding of App** Students continue to code app.
15 min	**Review and Demo** 1. Review the main components - Canvas, Ball, ImageSprite, Notifier. 2. Review the CT concepts - procedures, conditionals, variables, TinyDB. 3. Have students complete multiple choice questions. 4. Ask students who add features to their apps to demo them to their peers.

6. Assessment

Multiple-choice questions

1. A student is making a game app and uses the following code:
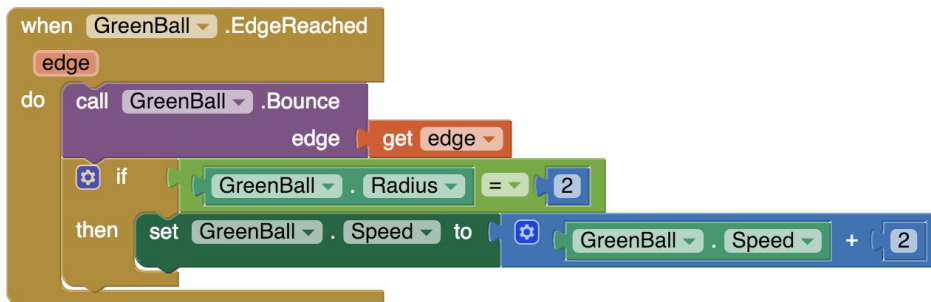
   What will happen when the GreenBall reaches an edge?

   A. The GreenBall will bounce off the edge only if the radius equals 2.

   B. The GreenBall will bounce off the edge and its speed will increase by 2.

   C. The GreenBall will bounce off the edge and its radius will increase by 2.

   D. The GreenBall will bounce off the edge and if the radius equals 2, its speed will increase by 2.

   Answer: D

2. What are benefits of making a procedure when programming?

   A. Code is organized according to a particular task.

   B. The same code can be used in multiple places by calling the procedure.

   C. If there is a change to code, it only has to be changed in one place.

   D. All of the above.

Answer: D

3. A programmer uses the following code blocks in a game app.
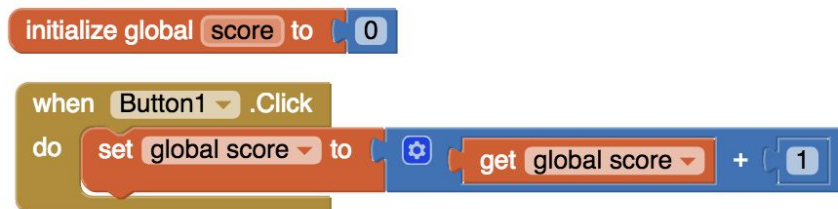
A user opens the app and clicks Button1 five times. What is the value of **score** then?

   A. 0
   B. 1
   C. 5
   D. 6

Answer: C

The user closes the app and re-opens it. What is the value of **count** then?

A. 0
B. 1
C. 5
D. 6

Answer: A

Survey of learning attitudes

In order to evaluate students' attitude, perception, and understanding towards coding, students are required to finish a 5-point scale survey below by putting a "✓" in the appropriate box.

After completion of this unit, I think…	Disagree	Somewhat disagree	Neutral	Somewhat agree	Agree
Learning how to make apps makes me want to learn more about coding.					
I feel more connected to the technology around me when I make apps.					
I am excited to share this app with friends and family.					

Self Assessment on Collaboration

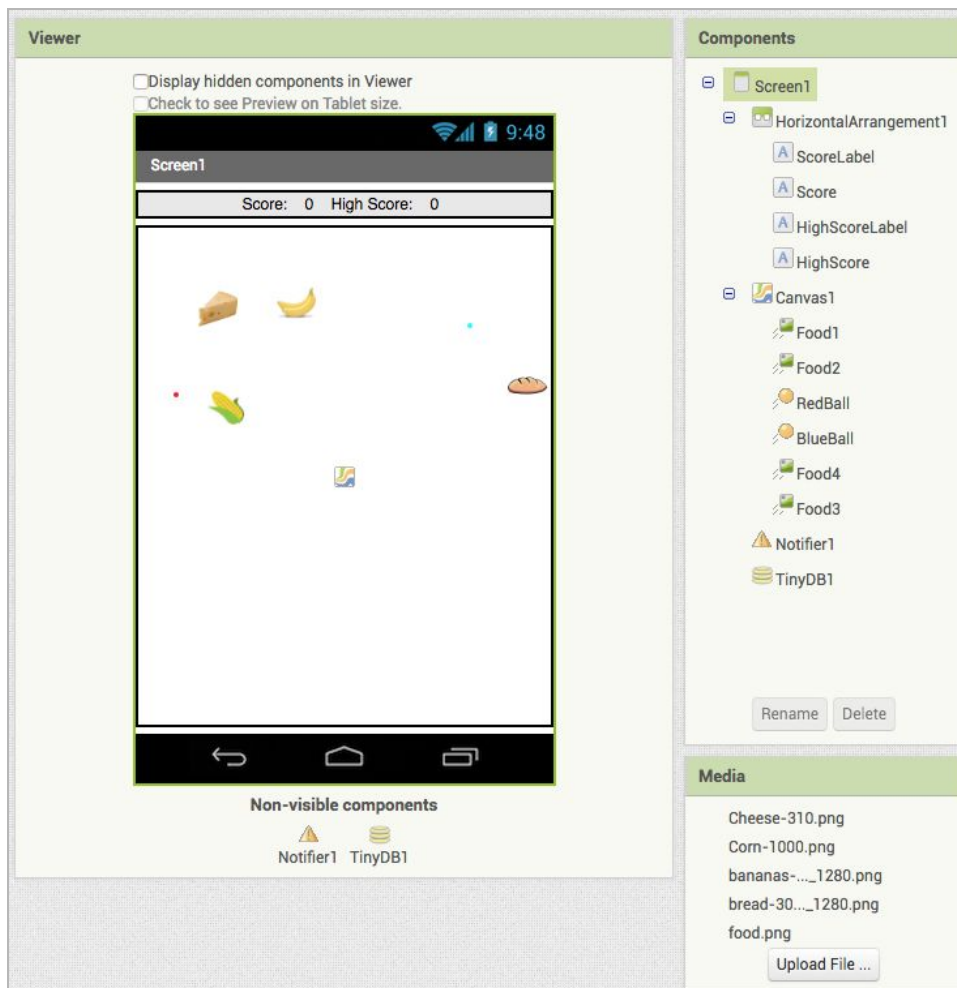Ask students to reflect on how well they worked with their partner, and to answer the following questions honestly.

1. Did you like working with another person? Do you feel you were a good partner, and respected and encouraged your partner in the project?

2. What was your role as a partner in this project? What did you do and what did your partner do?
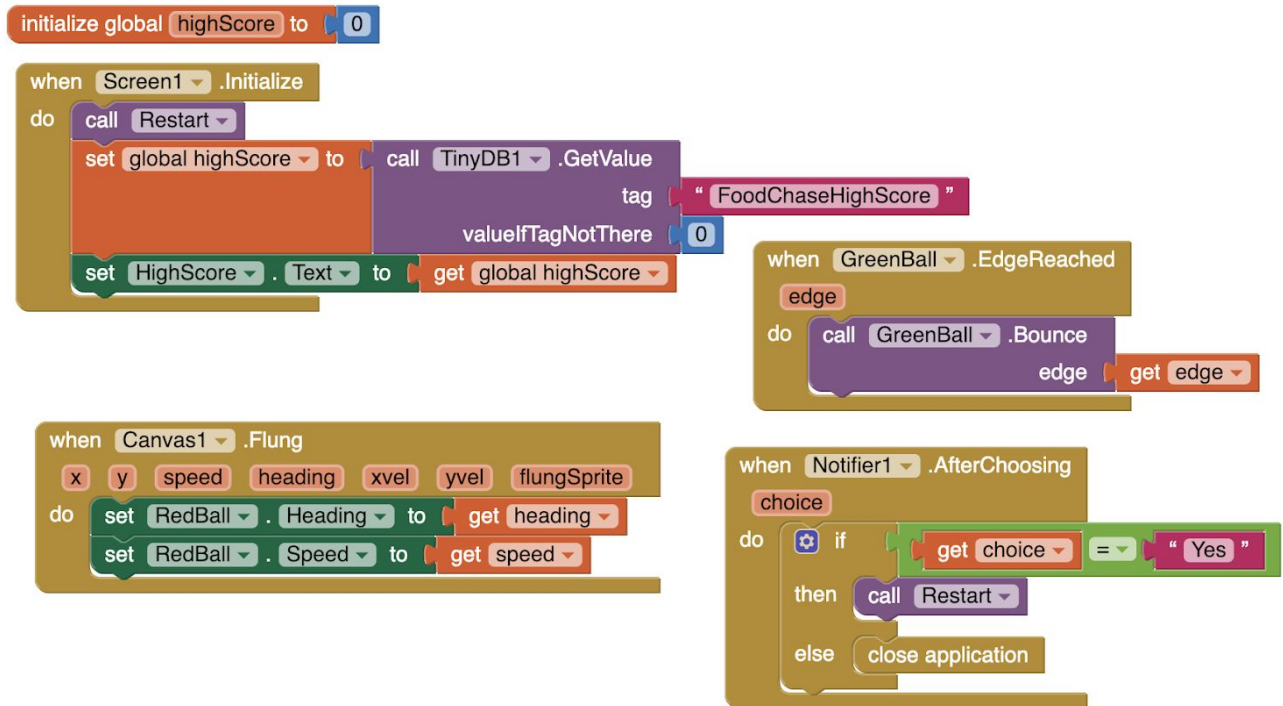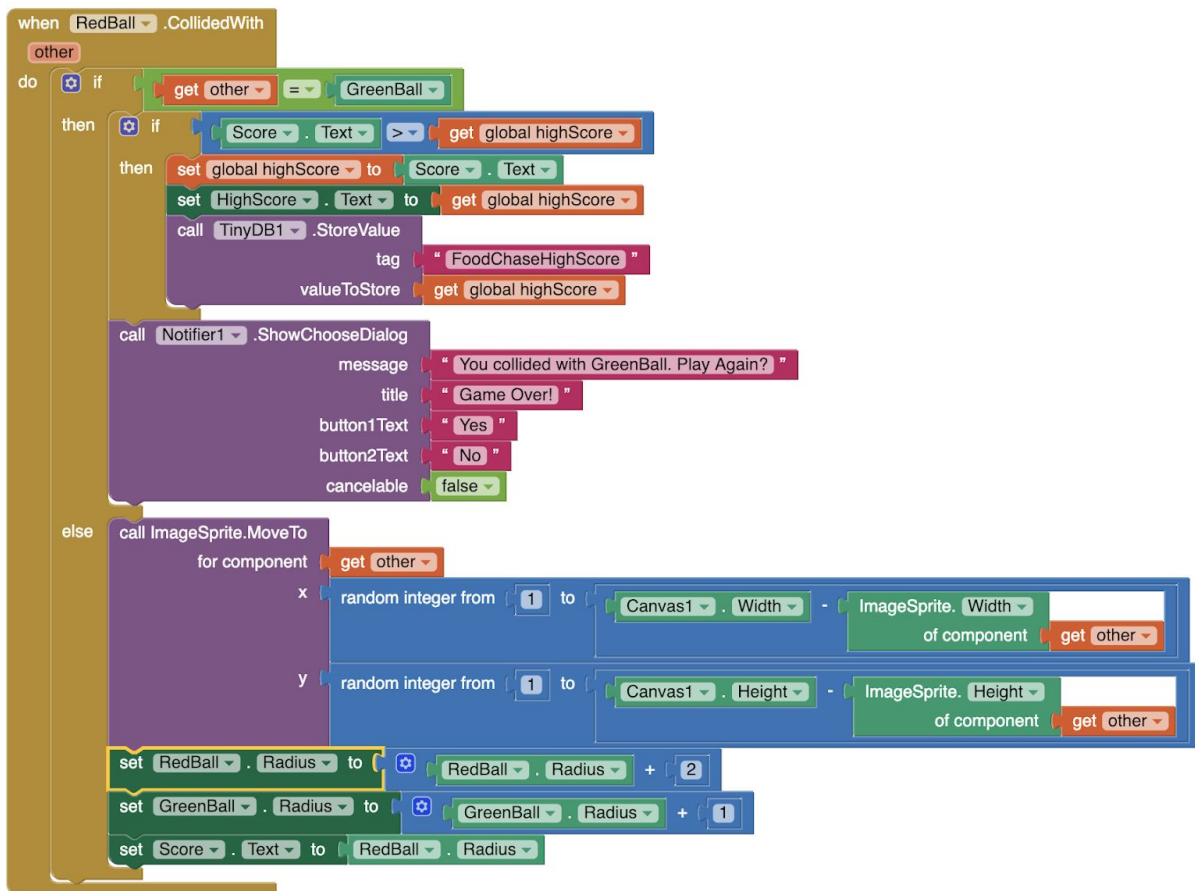
7. Screen Design and Code

Designer

Blocks

```
to Restart
do  call Food1 .MoveTo
                        x   random integer from  1  to   Canvas1 . Width  -  Food1 . Width
                        y   random integer from  1  to   Canvas1 . Height  -  Food1 . Height
    call Food2 .MoveTo
                        x   random integer from  1  to   Canvas1 . Width  -  Food2 . Width
                        y   random integer from  1  to   Canvas1 . Height  -  Food2 . Height
    call Food3 .MoveTo
                        x   random integer from  1  to   Canvas1 . Width  -  Food3 . Width
                        y   random integer from  1  to   Canvas1 . Height  -  Food3 . Height
    call Food4 .MoveTo
                        x   random integer from  1  to   Canvas1 . Width  -  Food4 . Width
                        y   random integer from  1  to   Canvas1 . Height  -  Food4 . Height
    call RedBall .MoveTo
                        x   random integer from  1  to  Canvas1 . Width
                        y   random integer from  1  to  Canvas1 . Height
    call GreenBall .MoveTo
                        x   random integer from  1  to  Canvas1 . Width
                        y   random integer from  1  to  Canvas1 . Height
    set RedBall . Radius to  2
    set GreenBall . Radius to  2
    set GreenBall . Speed to  5
    set GreenBall . Heading to  random integer from  1  to  360
    set Score . Text to  0
```

```
initialize global highScore to 0

when Screen1 .Initialize
do  call Restart
    set global highScore to   call TinyDB1 .GetValue
                                              tag   " FoodChaseHighScore "
                                valueIfTagNotThere   0
    set HighScore . Text to   get global highScore

when GreenBall .EdgeReached
    edge
do  call GreenBall .Bounce
                      edge   get edge

when Canvas1 .Flung
    x  y  speed  heading  xvel  yvel  flungSprite
do  set RedBall . Heading to   get heading
    set RedBall . Speed to   get speed

when Notifier1 .AfterChoosing
    choice
do  if   get choice = " Yes "
    then  call Restart
    else  close application
```

```
when  RedBall ▾ .CollidedWith
 other
do   ⚙ if        ┌ get other ▾  = ▾  ┌ GreenBall ▾
     then   ⚙ if        ┌ Score ▾ . Text ▾  > ▾  ┌ get global highScore ▾
            then   set global highScore ▾ to  ┌ Score ▾ . Text ▾
                   set HighScore ▾ . Text ▾ to  ┌ get global highScore ▾
                   call TinyDB1 ▾ .StoreValue
                                         tag  ┌ " FoodChaseHighScore "
                                  valueToStore  ┌ get global highScore ▾

                   call Notifier1 ▾ .ShowChooseDialog
                                      message  ┌ " You collided with GreenBall. Play Again? "
                                        title  ┌ " Game Over! "
                                   button1Text  ┌ " Yes "
                                   button2Text  ┌ " No "
                                    cancelable  ┌ false ▾

     else   call ImageSprite.MoveTo
                   for component  ┌ get other ▾
                              x  ┌ random integer from  ┌ 1  to  ┌ Canvas1 ▾ . Width ▾  -  ImageSprite. Width ▾
                                                                                                of component  ┌ get other ▾
                              y  ┌ random integer from  ┌ 1  to  ┌ Canvas1 ▾ . Height ▾  -  ImageSprite. Height ▾
                                                                                                of component  ┌ get other ▾
            set RedBall ▾ . Radius ▾ to  ┌ ⚙ ┌ RedBall ▾ . Radius ▾  +  ┌ 2
            set GreenBall ▾ . Radius ▾ to  ┌ ⚙ ┌ GreenBall ▾ . Radius ▾  +  ┌ 1
            set Score ▾ . Text ▾ to  ┌ RedBall ▾ . Radius ▾
```

Appendix 1
Teacher's Guide: Lesson 1

Learning Objectives

1.   Create a game app that uses Balls and ImageSprites on a Canvas.

2.   Collaborate using the Pair Programming model.

Lesson Outline

Introduction to Unit (10 minutes)

This will be the first unit where students make a game. Relate to games they've played on mobile devices.

1. Ask students if they enjoyed making the maze game app. Ask:

    a. How did the user control the ball movement in the maze game?

    b. What are some other ways you can control ball and sprite movement with a mobile app?

2. Explain that they will make a game app called "FoodChase" in this unit. It works as follows:

    a. User controls red ball by flinging action

    b. Red ball "eats" food to grow bigger

    c. Red ball avoids green ball, which moves randomly on screen

    d. If green and red balls collide, game is over

3. Demonstrate the finished app.

4. Explain that this unit will be a Pair Programming unit, so they will be assigned a partner

to work with. Remind the students of the driver/navigator model, and the rules below.

DO	DON'T
Be respectful Talk to one another about the work Explain what you are doing Think ahead and make suggestions Switch roles often	Be a bossy navigator Grab the driver's mouse or keyboard

Demonstration of Canvas, Ball and ImageSprite Components in App Inventor (10 minutes)

Introduce the new components students will use in this project. In the **Designer** window, open the **Drawing and Animation** drawer, and demonstrate the use of the three components: **Canvas**, **Ball**, and **ImageSprite**.

1. Drag out a **Canvas**, and change the *Width* and *Height* to "**Fill Parent**" so it fills the entire screen.
2. Drag out a **Ball**, and demonstrate changing the *PaintColor* and *Radius* to change how it looks.
3. Drag out an **ImageSprite**, and add a Picture so it appears on the **Canvas** in the Viewer. Demonstrate how to change the *Width* and *Height* to accommodate the size of the screen.
4. In general:
   a. **Canvas** is your background for sprites to appears and move.
   b. **Balls/ImageSprites** are the elements on the canvas that can be controlled by coding and user interaction.
5. Switch to the **Blocks Editor** and review some of the blocks used.
   a. **Canvas.Flung** event: for this app, transfer the speed and heading of the fling to set the Ball's movement
   b. **Ball.CollidedWith**: detects collision with other **Ball/ImageSprites**

c. **Ball.EdgeReached**: detects if **Ball** has touched an edge, and allows it to bounce off edge

Coding of App (20 minutes)

Student groups start to code Part 1 of app, following *Student Guide: Part 1*. Students will work using the Pair Programming model, so each student in a pair takes turns as driver/navigator.

Wrap-up (5 minute)

Check in with student groups to see they are making progress on the app. Ask if anyone is having problems. Explain that students will continue coding in the next lesson.

Appendix 2
Teacher's Guide: Lesson 2

Learning Objectives

1. Use a conditional properly to affect the flow of a program.
2. Demonstrate understanding of abstraction through the use of an Any Component block.
3. Work collaboratively to program, test, and debug a program.

Lesson Outline

Conditionals Review (10 minutes)

Remind students of the two types of conditional statements used so far.

The **if-then** block will test *if* a condition is true, and will execute the code in the *then* part of the block. If the condition is false, the code is skipped.

Most of the blocks that attach to the if block are found in the Logic (green) drawer, although you can also find similar blocks in the blue Math drawer (testing less than < and greater than > for example).

For example, in the **RedBall.CollidedWith** event block, if **RedBall** collides with **GreenBall**, notify the user the game is over.

You can also "mutate" the **if** block to make it **if-then-else** by clicking on the blue gear icon and then dragging **else** into the block. With **if-then-else**, you can specify two different sets of code - one for the true condition, and one for the false condition. For example, in **RedBall.CollidedWith**, the else condition would be collision with other ImageSprites, the Food.
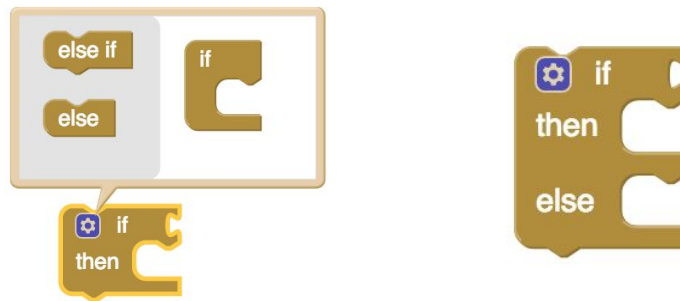
Any Component (5 minutes)

Any Component generalizes code blocks for a specific component type. The Any Component blocks appear at the bottom of the Blocks palette. Note there is an Any Component listed for each type of component within an app.

For this app, students will use Any ImageSprite to generalize the random placement of the Food **ImageSprites** in the **RedBall.CollidedWith** event block. Because the event block has **other** as an input parameter, we can use that to specify which Food **ImageSprite** we are talking about when we use **ImageSprite.MoveTo**.

The use of Any Component, and in this case, ImageSprite.MoveTo, is a demonstration of abstraction, an important Computational Thinking concept. Because the action
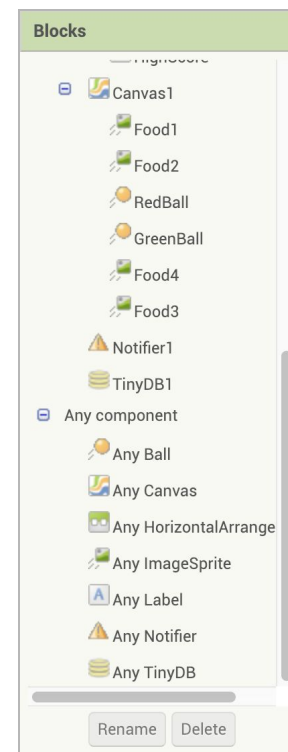
(MoveTo) for each Food ImageSprite is the same, we are generalizing, or abstracting the movement to work for any Food ImageSprite. Note that you can directly include the name of the ImageSprite, but in that case, you lose the power of the abstraction, because you would still need a separate block for each Food ImageSprite.
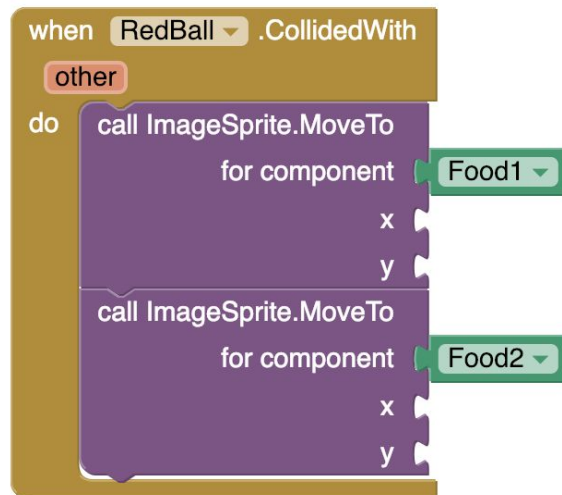
Here is the completed block. The benefit here is that you only need a single **ImageSprite.MoveTo** block, instead of separate **if** blocks to test for each of the five Food **ImageSprites**, each with its own separate **Food1.MoveTo** or **Food2.MoveTo** block, etc.
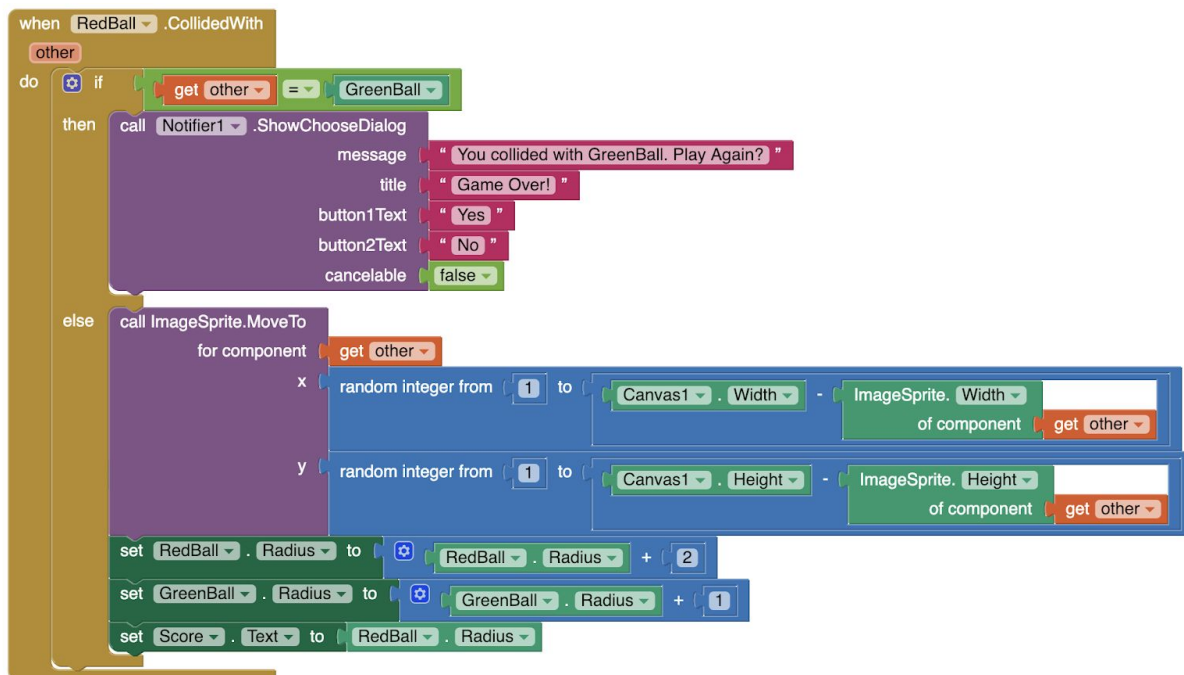
Note that in Lesson 3, in the **Restart** procedure, students will have to separately include **Food1.MoveT**o, **Food2.MoveTo** etc. blocks to randomly position them all. The reason is that we don't have the argument other (or another argument) to specify which component to move.

Students might question this. There is a way to generalize for **Restart**, but that requires creating a list for all the Food **ImageSprites**, which is more advanced for this point in the curriculum. Explain the need to specify the component in ImageSprite.MoveTo, which is available in RedBall.CollideWith's **other** parameter. **Restart** does not have that.

Coding of App (25 minutes)

Student groups continue working on the app. Depending on how far they got in Lesson 1, they may either continue with *Student Guide: Part 1*, or start *Student Guide: Part 2*.

Wrap-up (5 minutes)

Review conditionals and procedures. Check how far students have gotten with their apps.

Appendix 3
Teacher's Guide: Lesson 3

Learning Objectives

1. Demonstrate understanding of abstraction as it relates to the use of procedures in a program.
2. Demonstrate understanding of **if** and **if-then-else** conditional blocks.
3. Work collaboratively using the Pair Programming model.

Lesson Outline

Procedures (10 minutes)

Students will add a **Restart** procedure in Part 3 of the project. Review with students what a procedure is, how it is useful, and why it is helpful to use it in this project.

- Procedures are helpful to abstract and modularize code. Generally, it helps to separate out a particular "task" or action that may involve several code blocks. Then, the procedure can be "called" from other parts of the program.
- Procedures are helpful if you have similar code blocks in multiple parts of your program. Rather than have the same code in two places, it is easier and better practice to have one set of code blocks, organized as a procedure.
- Procedures also help with testing and debugging. Once you have tested a procedure and confirmed that it works correctly, then it does not have to be tested again, although it can

be called in more places.

- Procedures make updating code easier. If a procedure's blocks need to be altered or changed to accommodate a new feature, it only has to be changed in one place. In the alternate situation where you have multiple copies of the same code, the programmer must take care to to make the same change in all the different places the code exists.

Review Conditionals (5 minutes)

Go over the use of **if** and **if-then-else** blocks to allow for different actions within the app again. In this lesson, students will use **if-then-else** to determine the user's answer to the question to play again or quit when the two ball collide.

Coding of App (25 minutes)

Students continue working on the app. Depending on how far they got in Lesson 2, they may either continue *Student Guide: Part 2*, or do *Student Guide: Part 3*. Circulate to check each group's progress, and help out if there are any issues.

Wrap-up (5 minutes)

Check how far students have gotten with their apps. Review any issues or questions.

Appendix 4
Teacher's Guide: Lesson 4

Learning Objectives

1. Correctly use a variable in a program to store and retrieve information.
2. Use a flowchart to outline an algorithm.
3. Demonstrate understanding of persistence in a program, and use the App Inventor component TinyDB to store and retrieve data persistently.
4. Work collaboratively using the Pair Programming model in the classroom.

Introduction of Variables in App Inventor (10 minutes)

1. Explain variables by drawing boxes on the board. Each box is a storage box for a value. The contents of the box is the value. The value can change. The teacher can demonstrate this by erasing the value and changing it. Or by adding 1 to each value.
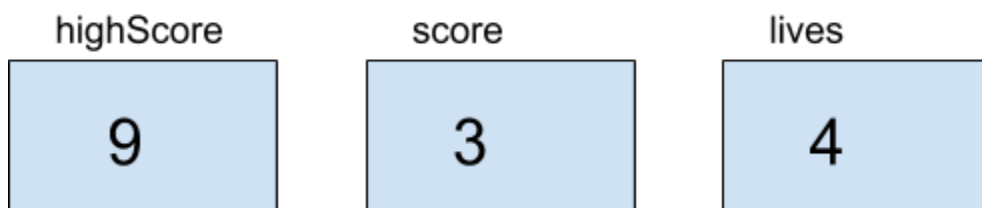

2. Demonstrate the Variables drawer in App Inventor, and the **initialize**, **get**, and **set** blocks.

Adding Variables and TinyDB (10 minutes)

Explain to students that **TinyDB** is a way to store data "persistently" for an app. When you use variables, they disappear when the app is closed. With **TinyDB**, you can store information that persists, even after the app is closed and reopened. **TinyDB** uses *tags* and *values*. A *tag* is like a variable name. A *value* is like the value that a variable has. **TinyDB.StoreValue** lets you store data in **TinyDB** by assigning it a tag. **TinyDB.GetValue** lets you retrieve that data, using the tag.

TinyDB is found in the **Storage** drawer in the **Designer** Palette.
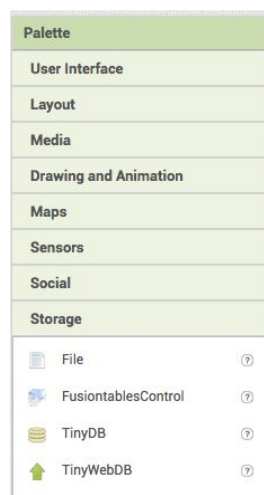
You can store values in and get values from **TinyDB**.

Tags are names given to variables. They help the user identify the values as they store and retrieve them from **TinyDB**.
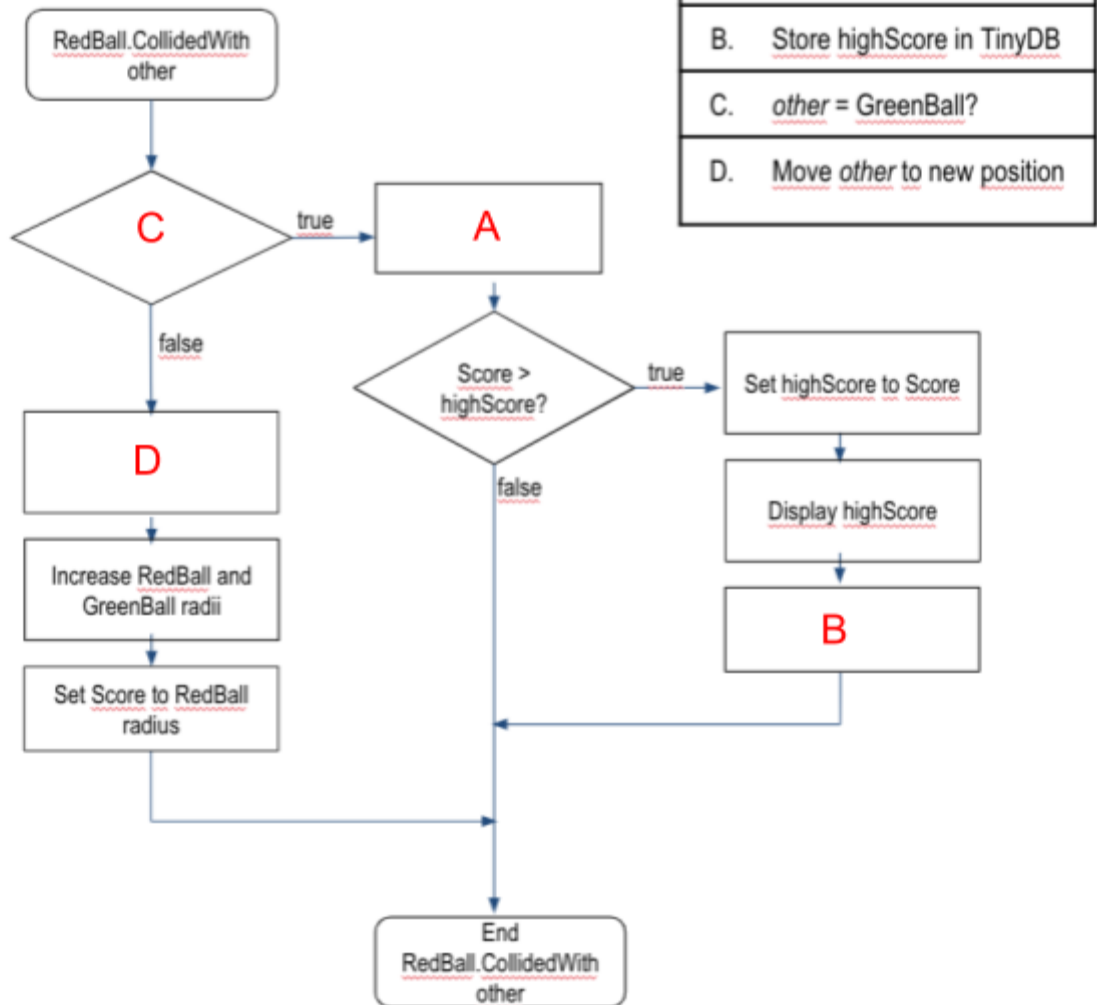
Comparing Variables to TinyDB

	Variable	**TinyDB**
Persistence	Anything stored in a variable is erased when the app closes	Anything stored in TinyDB with a tag can be retrieved at any point, even after the app is closed and reopened
Storing data	**set name** to value	**TinyDB.StoreValue** (tag, value)
Retrieving stored data	**get name**	**TinyDB.GetValue**(tag, valueIfTagNotThere)

Coding TinyDB (20 minutes)

1. Ask students to complete flowchart at start of Student Guide. Note, if students are doing the DIY version of the guide, they should complete it after they code the app. Answers are below.

With your partner, look at the following flowchart and fill in the missing blocks with the correct letter, according to this table.

A.	Notify user game is over
B.	Store highScore in TinyDB
C.	*other* = GreenBall?
D.	Move *other* to new position

2. Students complete *Student Guide: Part 4* to add variables and **TinyDB** to implement a high score for the game.

Wrap-up (5 minutes)

Review variables and **TinyDB**. Review the code for this lesson, and emphasize the differences between variables and stored values in **TinyDB**, in terms of how data persists.

Appendix 5
Teacher's Guide: Lesson 5

Learning Objectives

1. Test and debug an app to ensure it works correctly and completely.
2. Work collaboratively using the Pair Programming model in the classroom.

Introduction to Lesson (5 minutes)

Students may continue to work on their apps, if they still need time. For those who have completed Parts 1-4, they can try some of the challenges.

Coding of App (25 minutes)

Student groups continue to code the app. Depending on how far they've gotten, they can continue to work through the student guides. They can also add new features, using some of the suggestions at the end of the *Student Guide: Part 4*. They may also come up with their own ideas to make the game more interesting. Sound files are included in the unit documents, if students choose to add sounds to the game.

Review and Demo (15 minutes)

1. Review the main components - Canvas, Ball, ImageSprite, Notifier.
2. Review the CT concepts - procedures, conditionals, variables, TinyDB.
3. Have students complete multiple choice questions, survey of learning attitudes, and self assessment on collaboration.
4. Ask students who added features to their apps to demo them to their peers.