Sketch and Guess

1. Synopsis

In this unit, students will create a complex multiplayer drawing game, Sketch and Guess, in which a player can draw pictures and guess others' pictures, similar to the game of Pictionary. In the first lesson, students will develop the user interface and navigation between the three screens in the app. In the second lesson, they will add the ability to draw with the Canvas component. In the third lesson, students will add the ability to "send" drawings to other app users--so that they can watch the user draw in real time. In the fourth lesson, students will add the capability to guess the drawing. Some students may also add various features (such as multiple colors and changing line width) with the help of the *Student Guide: Challenge,* making the app more user-friendly and fun. Students will apply the computational thinking practice of being incremental and iterative, since they will modify and add complexity to the app over four lessons. They will increase their knowledge of the use of conditionals, as they will employ nested if statements. As always, testing and debugging will be necessary to make sure the apps work correctly.

2. Learning Objectives

After completing this unit, students will be able to:

1. Make a multiplayer drawing app that uses CloudDB.
2. Use CT concepts such as sequences, events, conditionals, parallelism, naming, operators, and data manipulation in creating an app.
3. Demonstrate understanding of how to use CloudDB to pass multiple pieces of information between devices;.
4. Work collaboratively to code and test a working multiplayer app.

3. Mapping with the Computational Thinking Framework

The following tables show the alignment of this unit with the intended learning outcomes of the computational thinking framework. The entries indicate the expected relevance of this unit to each outcome:

✔✔✔	:	High relevance
✔✔	:	Some relevance
✔	:	Low relevance

Computational Thinking Concepts

Sketch and Guess		
1. Sequences	✔✔✔	With added complexity and more code blocks, sequences become more critical to a working app.
2. Events	✔✔✔	Events used include Canvas.Dragged, CloudDB.DataChanged, Screen.Initialize, Spinner.AfterSelection.
3. Repetition		
4. Conditionals	✔✔	Conditionals are required to check which tags are being returned by CloudDB.
5. Parallelism	✔✔	Multiple students may play at one time, so parallelism occurs.
6. Naming	✔✔	Descriptive naming of variables, components, and tags in a complex app are necessary.
7. Operators		
8. Manipulation of data and elementary data structures	✔✔✔	This app requires list manipulation and CloudDB for data storage and communication.

Computational Thinking Practices

Sketch and Guess		
1. Reusing and remixing		
2. Being incremental and iterative	✔✔✔	Students will build the app incrementally over the five class periods.
3. Abstracting and modularizing		
4. Testing and debugging	✔✔✔	Students test over multiple devices with a partner.
5. Algorithmic thinking	✔✔✔	Algorithmic thinking is required to understand the complex functionality of drawing and sharing data between devices.

Computational Thinking Perspectives

Sketch and Guess		
1. Expressing	✔✔✔	Students can express themselves by building a drawing app.
2. Connecting	✔✔✔	Students connect with their real life by making a game they can play with friends.
3. Questioning	✔✔✔	Students will face challenges learning how to coordinate multiple players.
4. Computational identity	✔✔✔	Students build their confidence by building a complex game.
5. Digital empowerment	✔✔✔	Students feel empowered by building an app that can be played by many others.

4. Mapping with the CSTA Standards

This table shows the alignment of this unit with the intended learning outcomes to the CSTA CS Standards. The entries in the tables indicate the expected relevance of the unit to each outcome:

2-NI-04	Model the role of protocols in transmitting data across networks and the Internet. [C] NI: Network Communication & Organization [P] Abstraction (4.4)	Unplugged activity used to model CloudDB
2-DA-07	Represent data using multiple encoding schemes. [C] DA: Storage [P] Abstraction (all)	Tag/value pairs in CloudDB
2-AP-10	Use flowcharts and/or pseudocode to address complex problems as algorithms. [C] AP: Algorithms [P] Abstraction (4.4, 4.1)	student guides provide simple flowcharts for students to complete
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)	Boolean and number variables are used, as well as tag/value pairs in database.
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 if statements are used

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)	The unit is broken into 4 parts - students do not design themselves but incrementally build.
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)	Students use a template with some provided UI and code blocks.
2-AP-18	Systematically test and refine programs using a range of test cases. [C] AP: Program Development [P] Testing (6.1)	Testing happens within each lesson
2-IC-22	Collaborate with many contributors through strategies such as crowdsourcing or surveys when creating a computational artifact. [C] IC: Social Interactions [P] Collaborating (2.4), Creating (5.2)	Collaborative drawing

5. Learning Prerequisites

Students should have a command of the App Inventor development environment, and be familiar with CloudDB as a means of storing and sharing data in the cloud.

6. Lesson Plan

This unit consists of five 45 minutes lessons.

Lesson 1

Time	Activity
10 min	**Introduction to App, Adding Multiple Screens** 1. Demonstrate a simple app with multiple screens. (SketchandGuess_checkpoint1.aia). 2. Explain that students will create Screen1 along with a second screen, SketcherScreen, and the drawing will happen on that screen.
30 min	**Building Screens, Navigating Between Screens** 1. Ask students to pair up and work using the Pair Programming model to build part 1 of Sketch and Guess. They will build Screen1 and the SketcherScreen with the help of *Student Guide: Lesson 1*. 2. Test and debug the app.
5 min	**Wrap-up** Explain that students will add code to draw on the SketcherScreen in Lesson 2.

Lesson 2

Time	Activity
15 min	**Introduction to Drawing with Canvas Component** 1. Demonstrate Sketch and Guess with simple drawing capability (SketchandGuess_checkpoint2.aia). 2. Introduce drawing on a Canvas and how lines are drawn based on user input.
25 min	**Coding of Drawing Feature** 1. Ask students to pair up and work using the Pair Programming model to build part 2 of Sketch and Guess. They will code the drawing feature of Sketch and Guess with the help of *Student Guide: Lesson 2*. 2. Test and debug the app.
5 min	**Wrap-up** 1. Review Canvas and drawing features learned in this lesson. 2. Ask students: a. "Have you played any similar games before?" (Pictionary) b. "How can you make this app work like Pictionary?"

Lesson 3

Time	Activity
5 min	**Review of Drawing Feature** 1. Review the drawing feature added in Lesson 2. 2. Ask students how they can play this game with another person on a single device. 3. Ask students, "How can you play this game with partners who are using different devices?" 4. Teacher demonstrates the CloudDB version of the drawing app (SketchandGuess_checkpoint3.aia) to show Sketch and Guess between two devices.
10 min	**Demonstrating Drawing Using CloudDB** Run unplugged activity with students acting as CloudDB, and Sketcher and Guesser.
25 min	**Adding CloudDB Component to the App** 1. Explain that whenever a user draws something on their device, they will also save the x,y coordinates for the start and end of the drag event to CloudDB. Other users will take that information and use it to draw the same line on their device. 2. Ask students to work using the Pair Programming model, following *Student Guide: Lesson 3*, to add CloudDB to their apps. 3. Ask students to try the app with their partner. Each student will download the apk to their own tablet and can play the app together. One student draws and their partner can see the drawing happen on their tablet too.
5 min	**Wrap-up** Ask students how they might make this into a Sketch and Guess game, where one person draws and other players guess.

Lesson 4

Time	Activity
10 min	**Review and Introduction to Lesson** 1. Ask the whole class what else can be improved, and how they might make this into a Sketch and Guess game, where one person draws and other players guess. 2. The teacher demonstrates and explains the new function of the game: One person sketches the randomly displayed object. Other players may watch the drawing and guess what is being drawn. If they guess correctly, they receive a message and all players are notified. (SketchandGuess_complete.aia) 3. Show students how to add a Spinner component, and demonstrate how it works.
30 min	**Coding the Sketch and Guess App** Following *Student Guide: Lesson 4*, students add code to their apps to make a working "Sketch and Guess" game. If students complete Part 4, they may follow *Student Guide: Challenge* to add color and pen size choice to the game.
5 min	**Wrap-up** 1. Check in with students to see how much of the app they have completed. 2. Explain the Challenge features that can be added to the game.

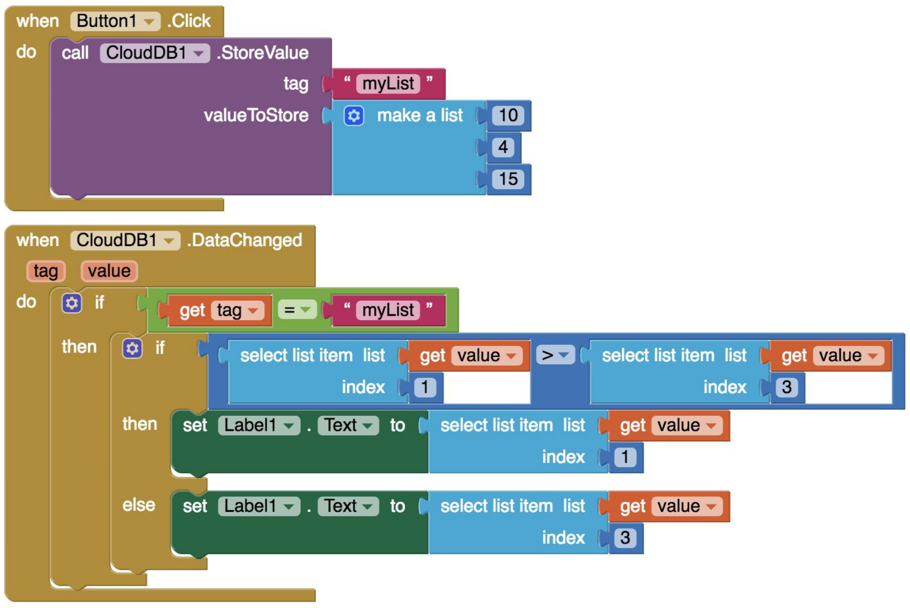
Lesson 5

Time	Activity
5 min	**Introduction to Lesson** 1. Ask students if there are other features they would like to add to the Sketch and Guess app. 2. Students can either finish the app if they have not completed it, or they can try the Challenge, where they can add color buttons and a slider to change the line width in the app.
25 min	**Coding** 1. Students who have not completed Parts 1-4 may work to complete the standard app. 2. Students may try the Challenge. 3. Students may consider adding their own new features, with teacher approval.
15 min	**Wrap-up** 1. Review the use of CloudDB in this unit. 2. Ask students to reflect on the game, and ask for volunteers to share any new features added. 3. Ask students to answer multiple choice questions and learning attitudes survey.

7. Assessment

Multiple Choice Questions

1. If the following code is used, and the user presses Button1, what is displayed in Label1?



```
when Button1 .Click
do  call CloudDB1 .StoreValue
                        tag  " myList "
               valueToStore  make a list  10
                                          4
                                          15

when CloudDB1 .DataChanged
  tag  value
do  if  get tag = " myList "
    then  if  select list item  list  get value   >  select list item  list  get value
                               index  1                             index  3
          then  set Label1 . Text to  select list item  list  get value
                                                       index  1
          else  set Label1 . Text to  select list item  list  get value
                                                       index  3
```

A. myList

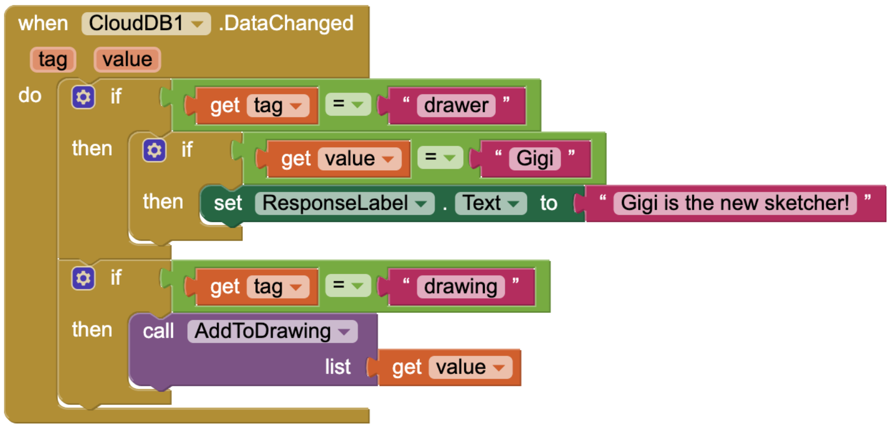
B. 10

C. 4

D. 15

Answer: D

2. If the following code is used and the tag is “drawer” and value is “Henry”, what will happen in the app?



```
when CloudDB1 .DataChanged
  tag  value
do  if  get tag = "drawer"
    then  if  get value = "Gigi"
          then  set ResponseLabel . Text to "Gigi is the new sketcher!"
    if  get tag = "drawing"
    then  call AddToDrawing
                      list  get value
```

A. “Gigi is the new drawer!” will be displayed.

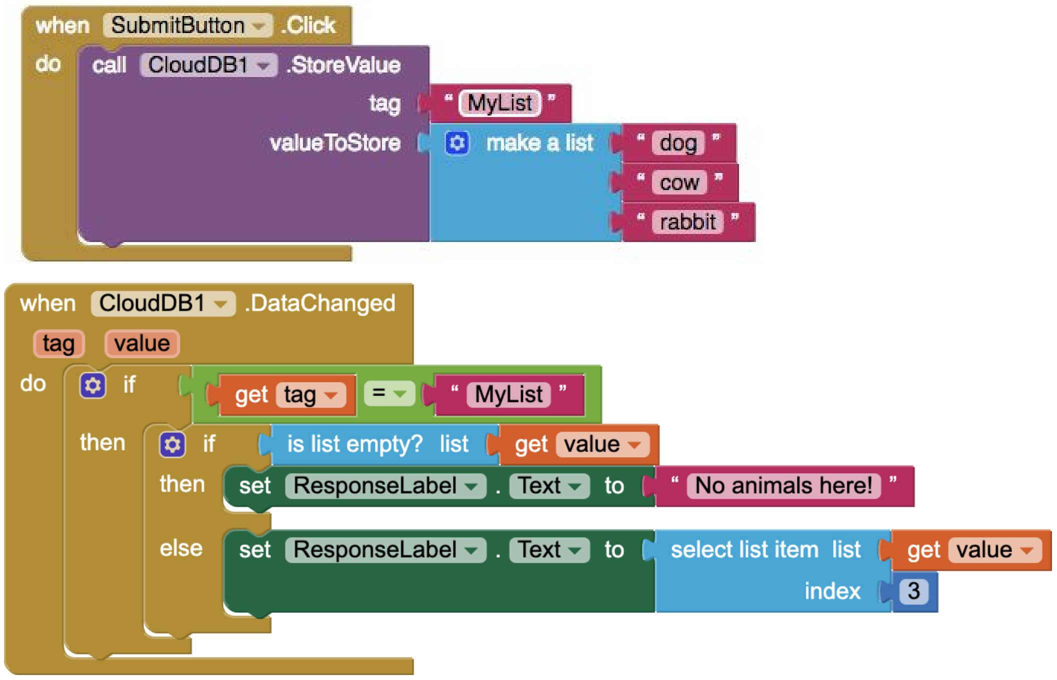
B. “Henry is the new drawer!” will be displayed.

C. The AddToDrawing procedure is called.

D. Nothing.

Answer: D

3. The following code is used in an app. The user presses the Submit button. What should happen?



```
when SubmitButton .Click
do  call CloudDB1 .StoreValue
                      tag  " MyList "
              valueToStore  make a list  " dog "
                                         " cow "
                                         " rabbit "

when CloudDB1 .DataChanged
  tag  value
do  if  get tag = " MyList "
    then  if  is list empty? list  get value
          then  set ResponseLabel . Text to  " No animals here! "
          else  set ResponseLabel . Text to  select list item  list  get value
                                                               index  3
```

A. “dog” is displayed.

B. “cow” is displayed.

C. “rabbit” is displayed.

D. “No animals here” is displayed.

E. Nothing.

(Answer: C)

Survey of learning attitudes

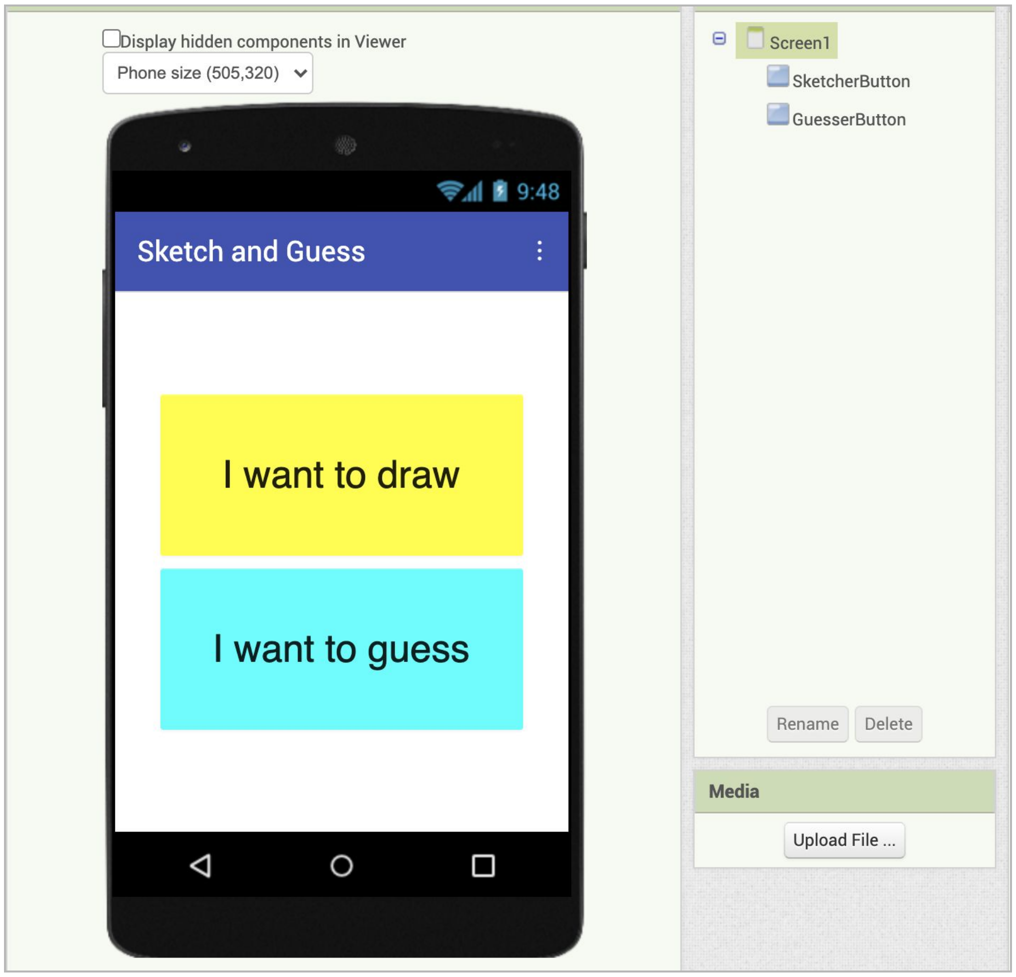
In order to evaluate students' attitude, perception, and understanding towards coding, students are required to finish a 5-point scale survey below by putting a "✓" in the appropriate box.

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.					

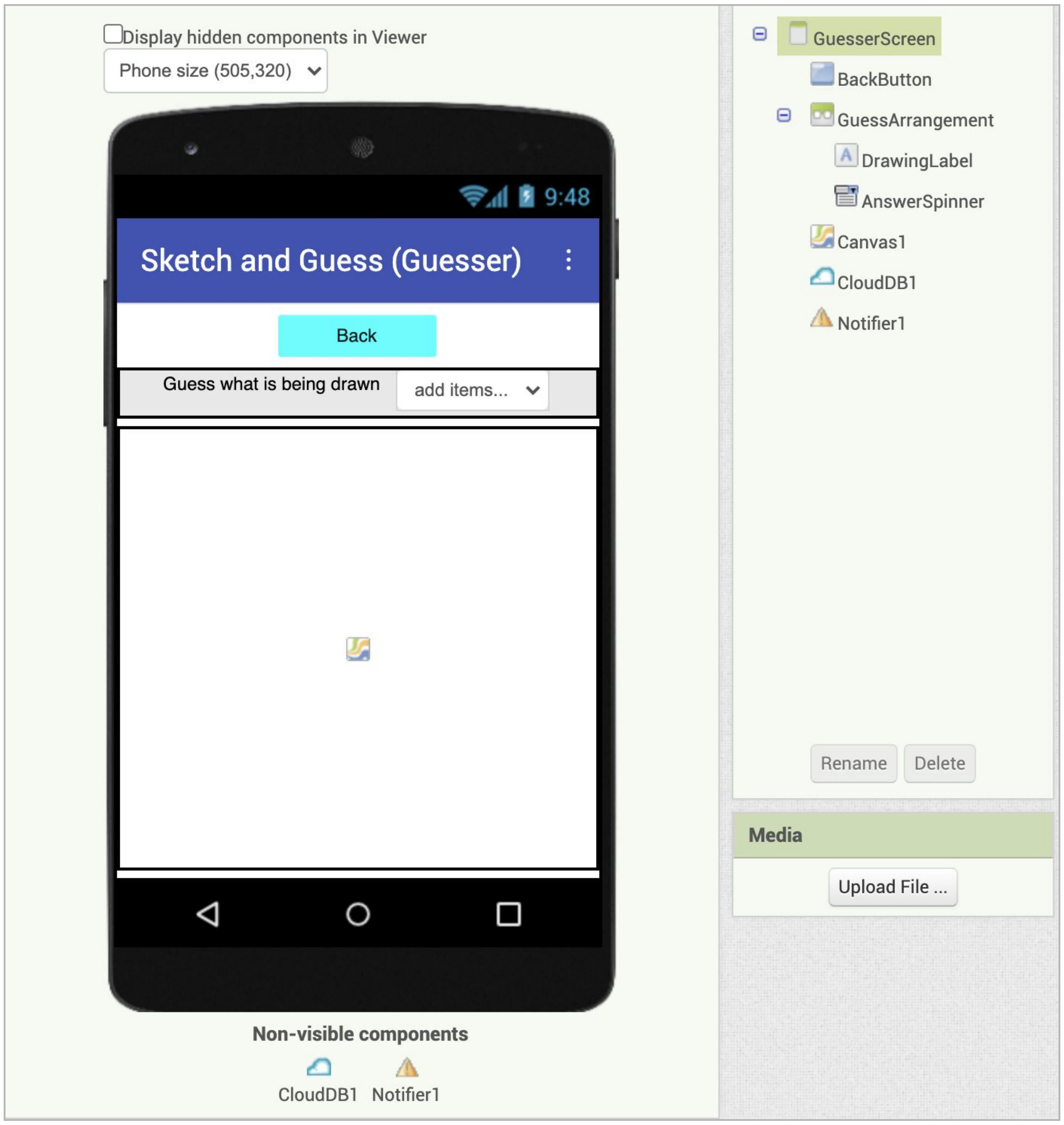
8. Screen Design and Code

Designer

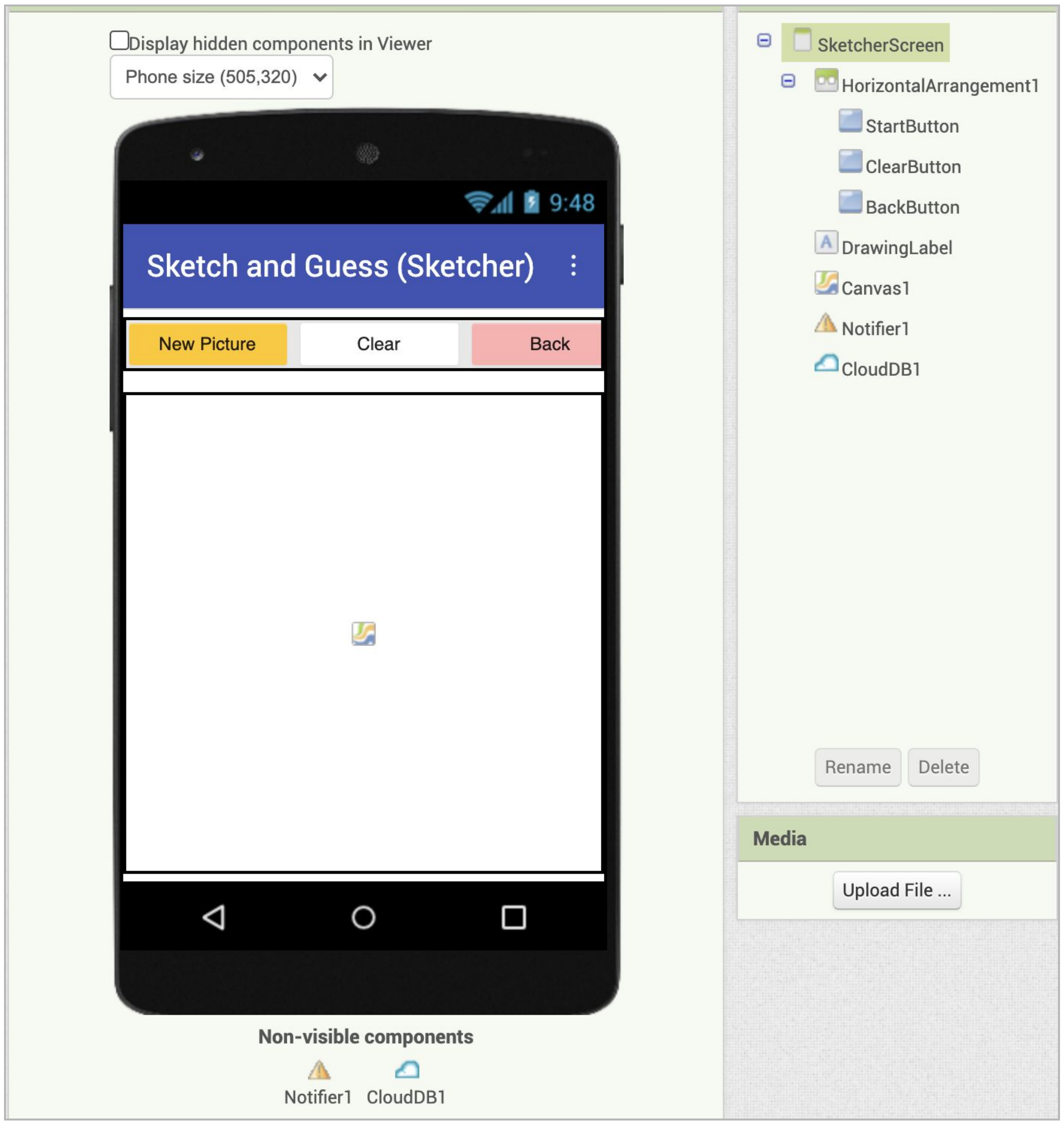
Screen1



GuesserScreen



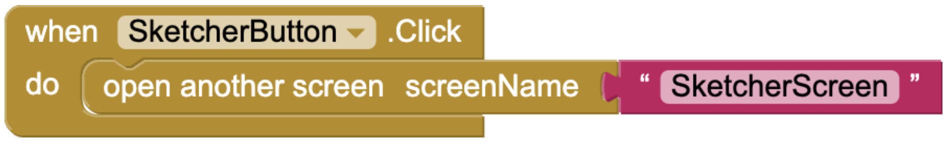
SketcherScreen



Blocks

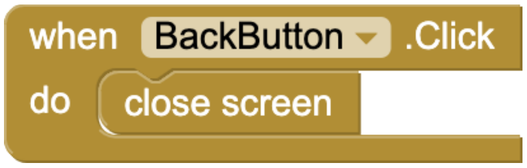
Checkpoint1

Screen1



```
when SketcherButton .Click
do   open another screen screenName " SketcherScreen "
```

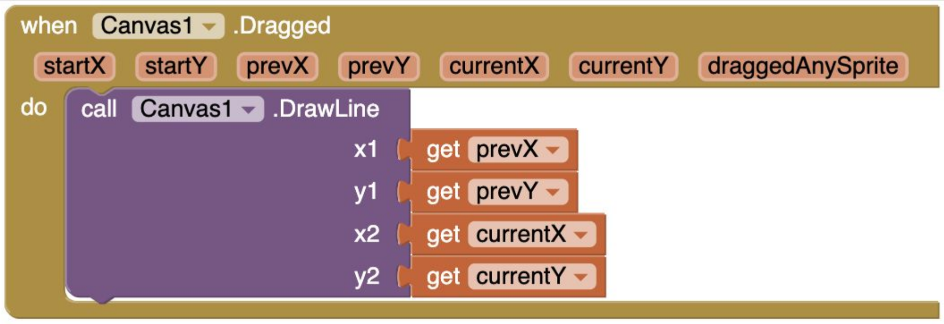
SketcherScreen



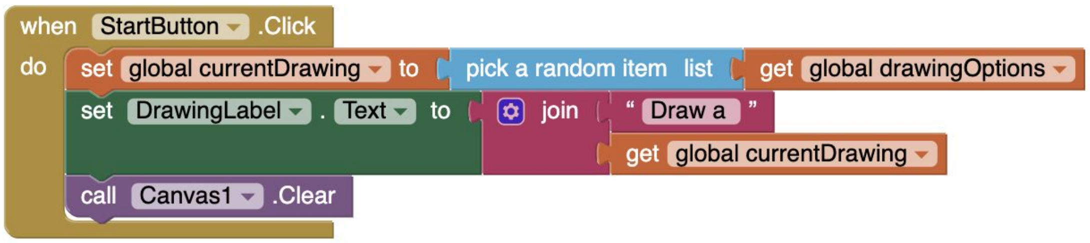
```
when BackButton .Click
do   close screen
```

Checkpoint 2 (added to above blocks)

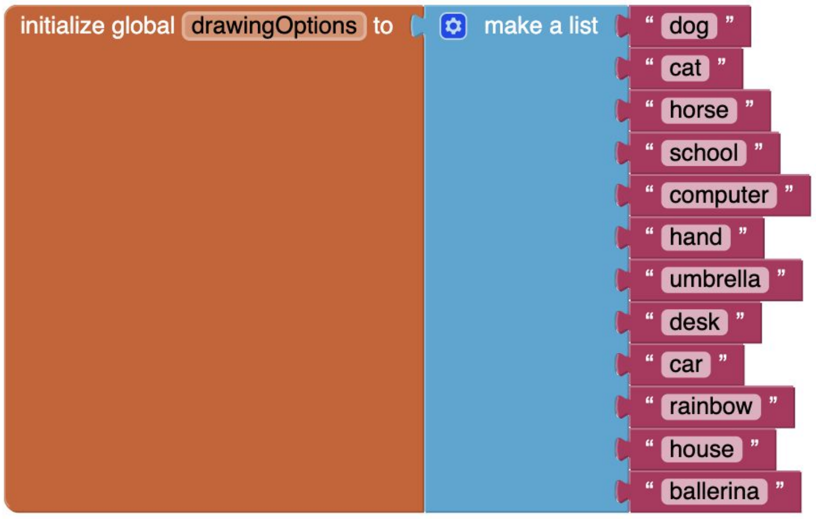
SketcherScreen



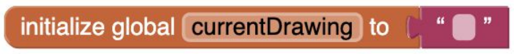
```
when Canvas1 .Dragged
  startX  startY  prevX  prevY  currentX  currentY  draggedAnySprite
do  call Canvas1 .DrawLine
                        x1  get prevX
                        y1  get prevY
                        x2  get currentX
                        y2  get currentY
```



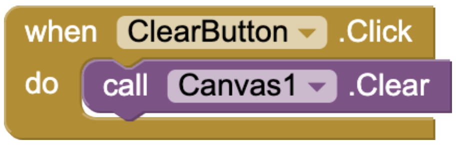
```
when StartButton .Click
do  set global currentDrawing to  pick a random item  list  get global drawingOptions
    set DrawingLabel . Text to  join  " Draw a "
                                      get global currentDrawing
    call Canvas1 .Clear
```



```
initialize global drawingOptions to  make a list  " dog "
                                                  " cat "
                                                  " horse "
                                                  " school "
                                                  " computer "
                                                  " hand "
                                                  " umbrella "
                                                  " desk "
                                                  " car "
                                                  " rainbow "
                                                  " house "
                                                  " ballerina "
```



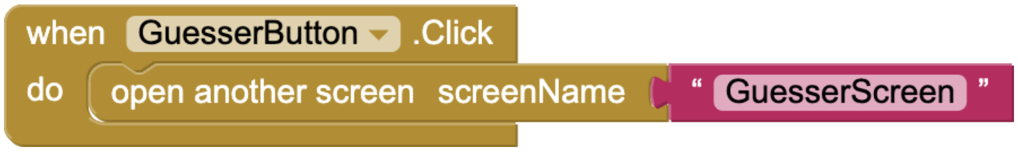
```
initialize global currentDrawing to  " "
```



```
when ClearButton .Click
do  call Canvas1 .Clear
```

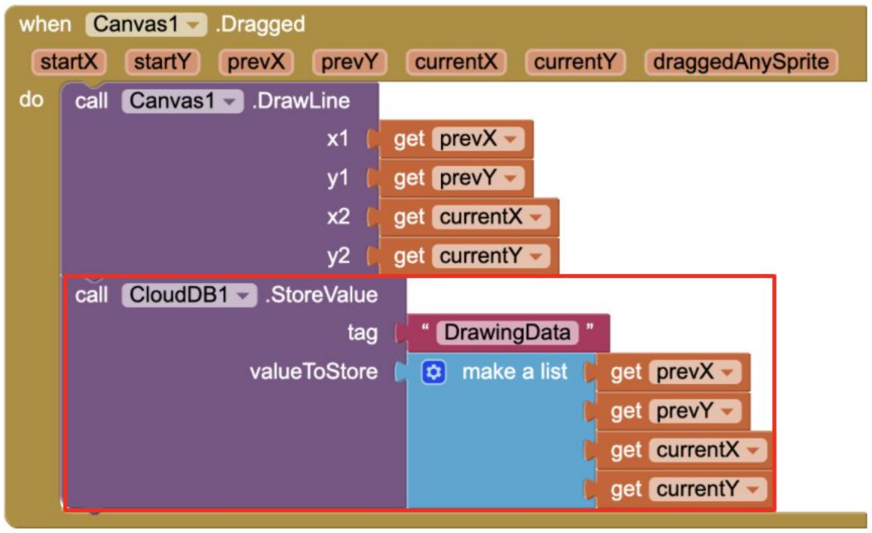
Checkpoint 3 (added to above blocks - blocks outlined in red are added to existing events)

Screen1

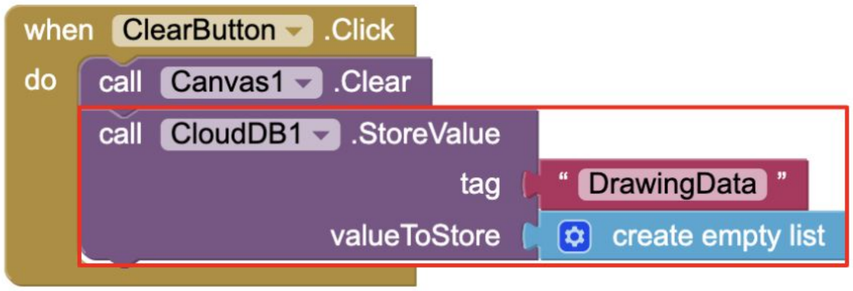


```
when GuesserButton .Click
do  open another screen  screenName  " GuesserScreen "
```

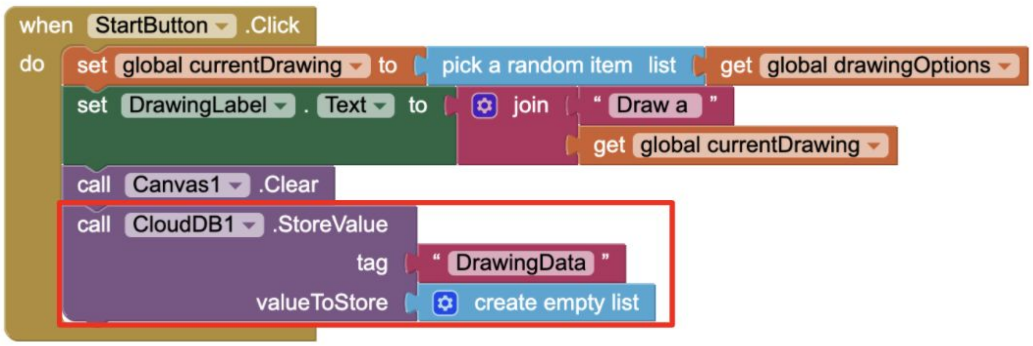
SketcherScreen



```
when Canvas1 .Dragged
  startX  startY  prevX  prevY  currentX  currentY  draggedAnySprite
do  call Canvas1 .DrawLine
                   x1  get prevX
                   y1  get prevY
                   x2  get currentX
                   y2  get currentY
    call CloudDB1 .StoreValue
                   tag  " DrawingData "
           valueToStore  make a list  get prevX
                                      get prevY
                                      get currentX
                                      get currentY
```

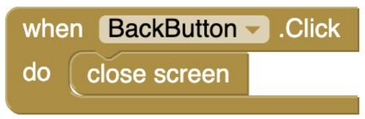


```
when ClearButton .Click
do  call Canvas1 .Clear
    call CloudDB1 .StoreValue
                   tag  " DrawingData "
           valueToStore  create empty list
```

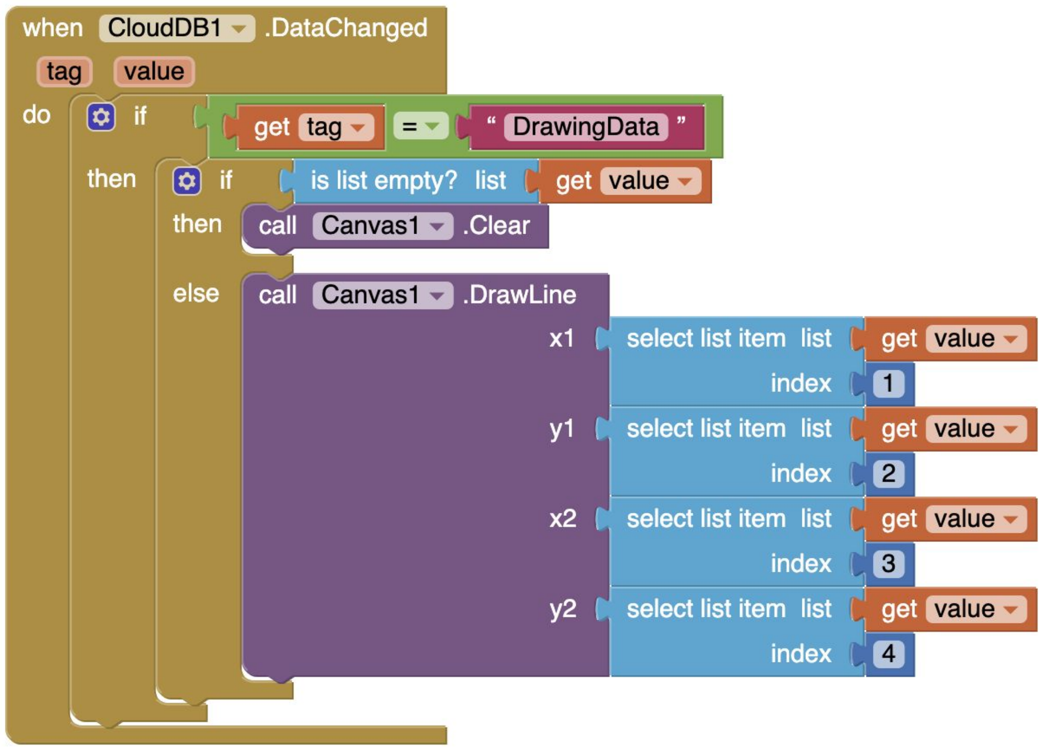


```
when StartButton .Click
do  set global currentDrawing to  pick a random item  list  get global drawingOptions
    set DrawingLabel . Text to  join  " Draw a "
                                      get global currentDrawing
    call Canvas1 .Clear
    call CloudDB1 .StoreValue
                   tag  " DrawingData "
           valueToStore  create empty list
```

GuesserScreen



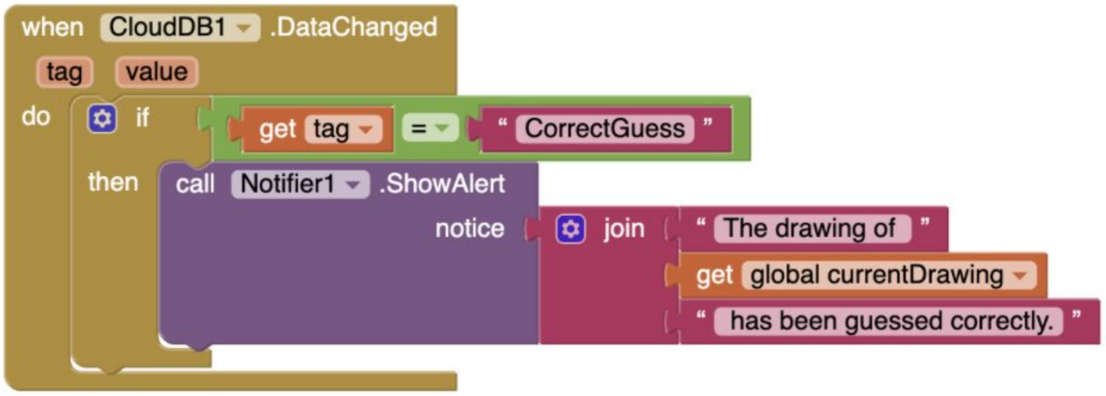
```
when BackButton .Click
do  close screen
```



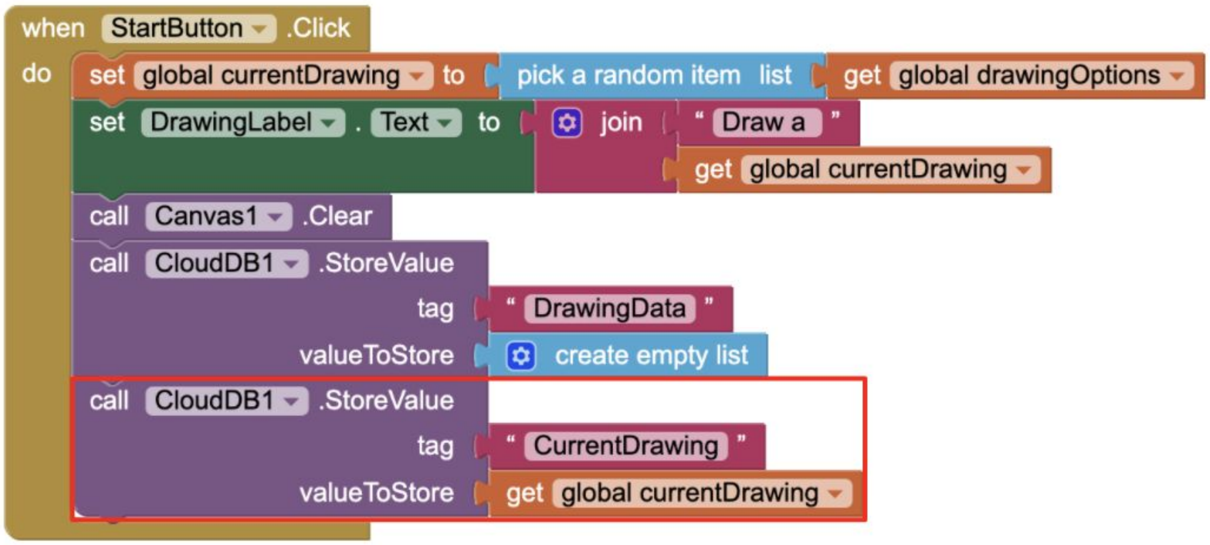
```
when CloudDB1 .DataChanged
  tag  value
do  if  get tag = " DrawingData "
    then  if  is list empty? list  get value
          then  call Canvas1 .Clear
          else  call Canvas1 .DrawLine
                           x1  select list item  list  get value
                                                 index 1
                           y1  select list item  list  get value
                                                 index 2
                           x2  select list item  list  get value
                                                 index 3
                           y2  select list item  list  get value
                                                 index 4
```

Complete: (added to above blocks - blocks outlined in red are added to existing events)

SketcherScreen

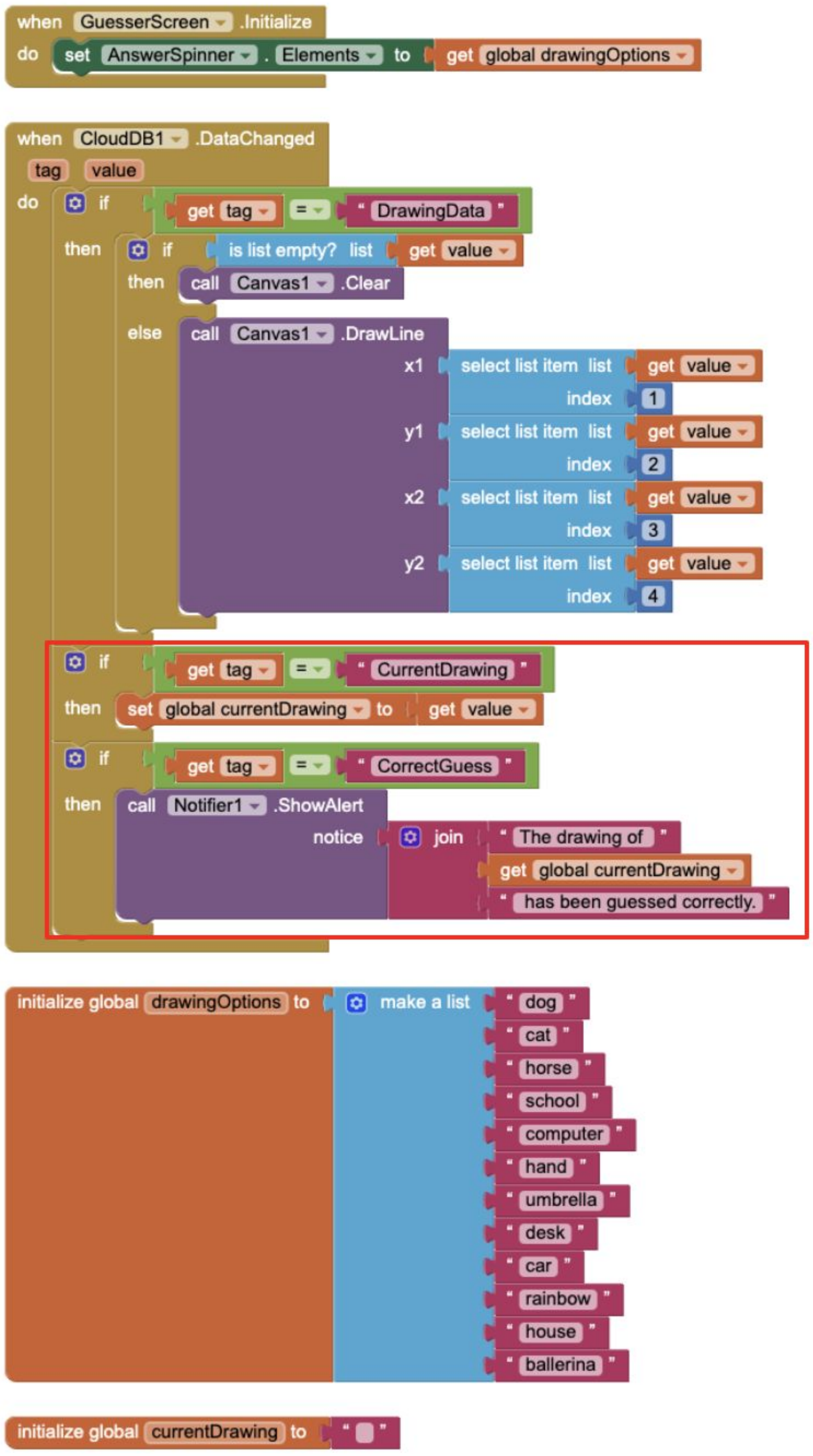


```
when CloudDB1 .DataChanged
  tag  value
do  if  get tag = " CorrectGuess "
    then  call Notifier1 .ShowAlert
                notice  join  " The drawing of "
                              get global currentDrawing
                              " has been guessed correctly. "
```

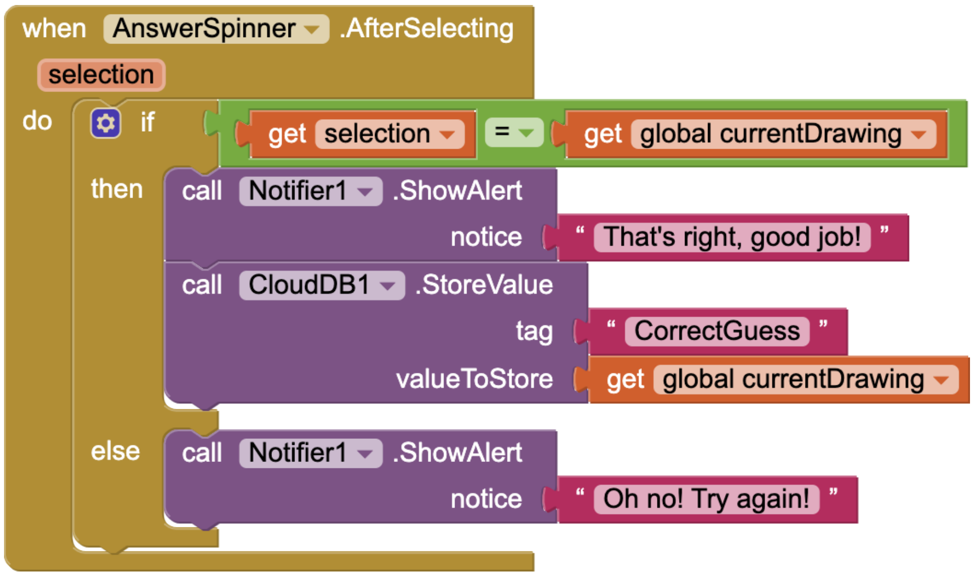


```
when StartButton .Click
do  set global currentDrawing to  pick a random item  list  get global drawingOptions
    set DrawingLabel . Text to  join  " Draw a "
                                      get global currentDrawing
    call Canvas1 .Clear
    call CloudDB1 .StoreValue
                 tag  " DrawingData "
        valueToStore  create empty list
    call CloudDB1 .StoreValue
                 tag  " CurrentDrawing "
        valueToStore  get global currentDrawing
```

GuesserScreen



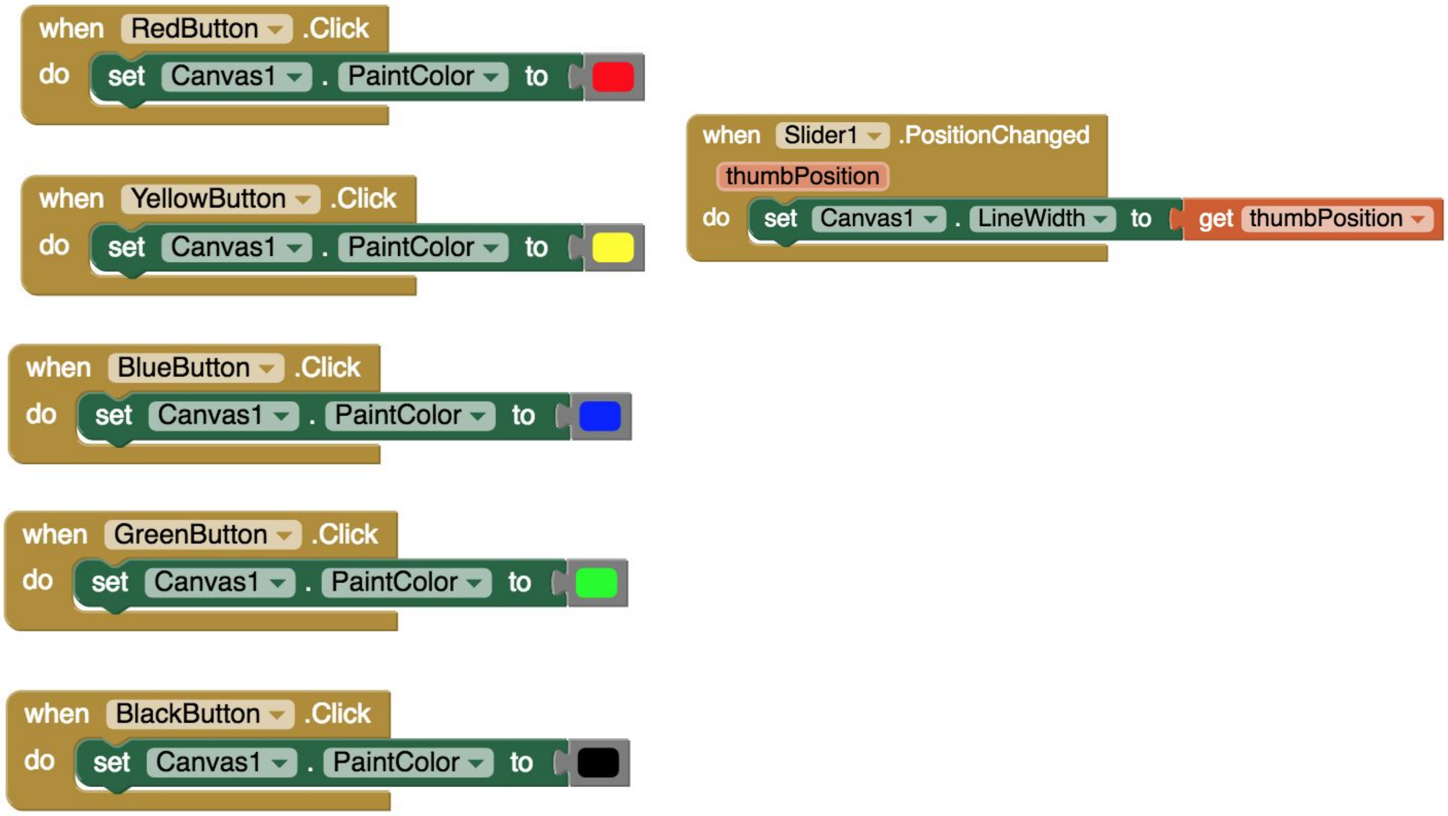
```
when GuesserScreen.Initialize
do  set AnswerSpinner.Elements to get global drawingOptions

when CloudDB1.DataChanged
  tag  value
do  if  get tag = "DrawingData"
    then  if  is list empty? list  get value
          then  call Canvas1.Clear
          else  call Canvas1.DrawLine
                      x1  select list item  list  get value
                                            index 1
                      y1  select list item  list  get value
                                            index 2
                      x2  select list item  list  get value
                                            index 3
                      y2  select list item  list  get value
                                            index 4
    if  get tag = "CurrentDrawing"
    then  set global currentDrawing to  get value
    if  get tag = "CorrectGuess"
    then  call Notifier1.ShowAlert
                notice  join  "The drawing of"
                              get global currentDrawing
                              "has been guessed correctly."

initialize global drawingOptions to  make a list  "dog"
                                                  "cat"
                                                  "horse"
                                                  "school"
                                                  "computer"
                                                  "hand"
                                                  "umbrella"
                                                  "desk"
                                                  "car"
                                                  "rainbow"
                                                  "house"
                                                  "ballerina"

initialize global currentDrawing to  ""
```



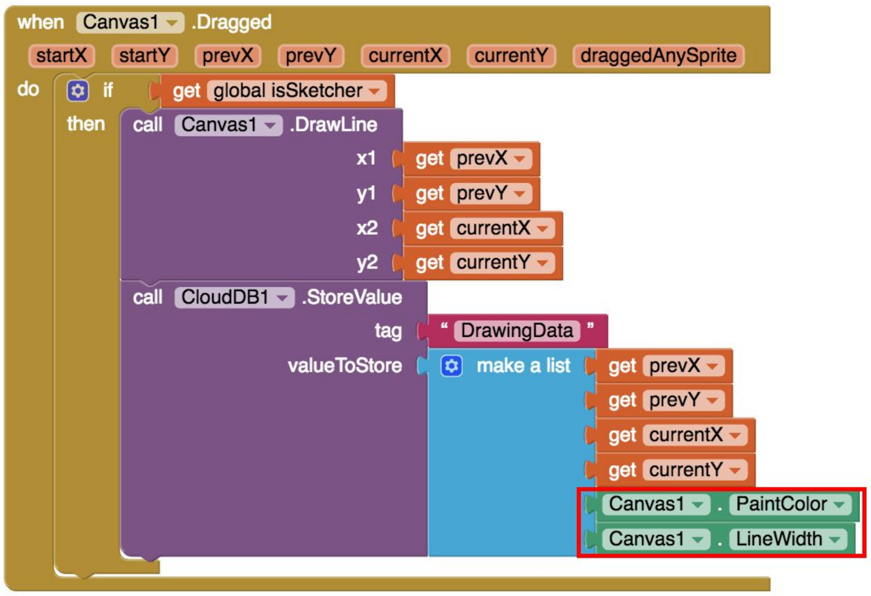
```
when AnswerSpinner .AfterSelecting
  selection
do  if  get selection = get global currentDrawing
    then  call Notifier1 .ShowAlert
                  notice  " That's right, good job! "
          call CloudDB1 .StoreValue
                  tag  " CorrectGuess "
                  valueToStore  get global currentDrawing
    else  call Notifier1 .ShowAlert
                  notice  " Oh no! Try again! "
```

Challenge: (added to above blocks - blocks outlined in red are added to existing events)

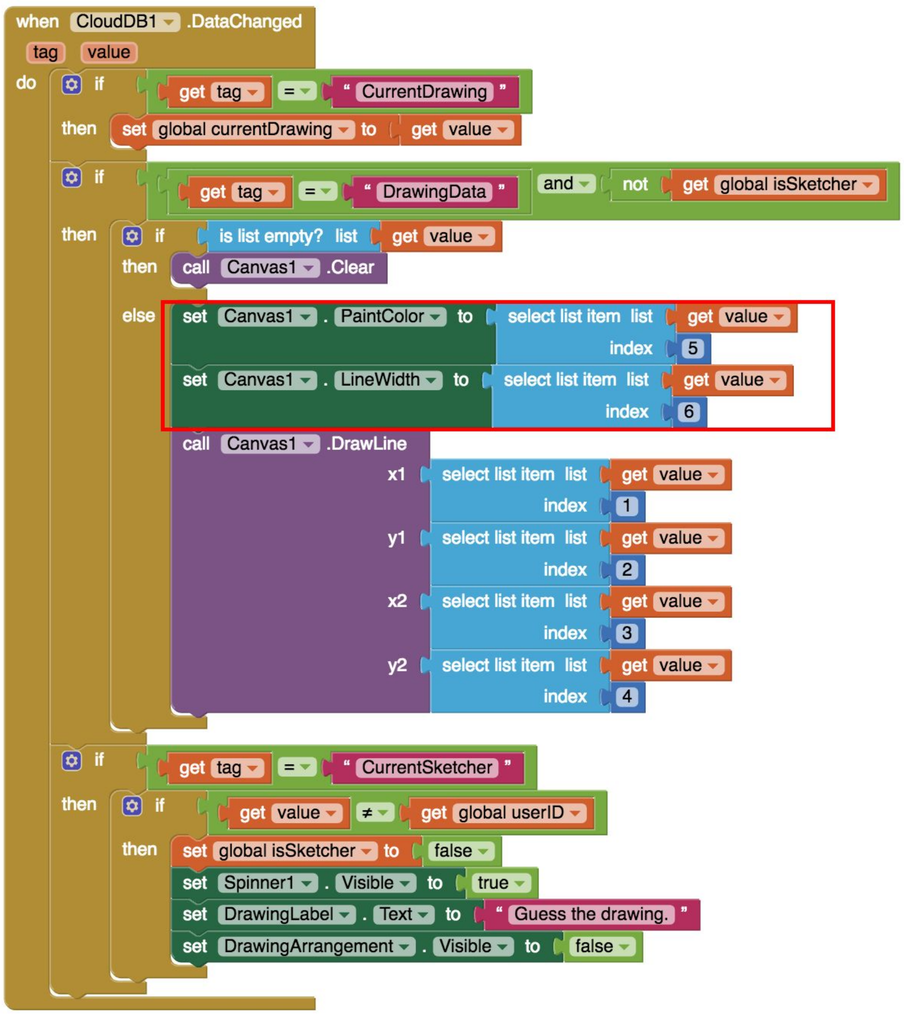
SketcherScreen



```
when RedButton .Click
do  set Canvas1 . PaintColor to 🟥

when YellowButton .Click
do  set Canvas1 . PaintColor to 🟨

when BlueButton .Click
do  set Canvas1 . PaintColor to 🟦

when GreenButton .Click
do  set Canvas1 . PaintColor to 🟩

when BlackButton .Click
do  set Canvas1 . PaintColor to ⬛

when Slider1 .PositionChanged
  thumbPosition
do  set Canvas1 . LineWidth to get thumbPosition
```



```
when Canvas1 .Dragged
  startX  startY  prevX  prevY  currentX  currentY  draggedAnySprite
do  if  get global isSketcher
    then  call Canvas1 .DrawLine
                 x1  get prevX
                 y1  get prevY
                 x2  get currentX
                 y2  get currentY
          call CloudDB1 .StoreValue
                 tag  " DrawingData "
                 valueToStore  make a list  get prevX
                                            get prevY
                                            get currentX
                                            get currentY
                                            Canvas1 . PaintColor
                                            Canvas1 . LineWidth
```



```
when CloudDB1 .DataChanged
  tag  value
do  if  get tag = " CurrentDrawing "
    then  set global currentDrawing to  get value
    if  get tag = " DrawingData "  and  not  get global isSketcher
    then  if  is list empty?  list  get value
          then  call Canvas1 .Clear
          else  set Canvas1 . PaintColor to  select list item  list  get value
                                                               index  5
                set Canvas1 . LineWidth to  select list item  list  get value
                                                              index  6
                call Canvas1 .DrawLine
                       x1  select list item  list  get value
                                             index  1
                       y1  select list item  list  get value
                                             index  2
                       x2  select list item  list  get value
                                             index  3
                       y2  select list item  list  get value
                                             index  4
    if  get tag = " CurrentSketcher "
    then  if  get value ≠ get global userID
          then  set global isSketcher to  false
                set Spinner1 . Visible to  true
                set DrawingLabel . Text to  " Guess the drawing. "
                set DrawingArrangement . Visible to  false
```

Appendix 1
Unit 9 Teacher's Guide: Lesson 1

Learning Objectives

At the end of this lesson, students should be able to:

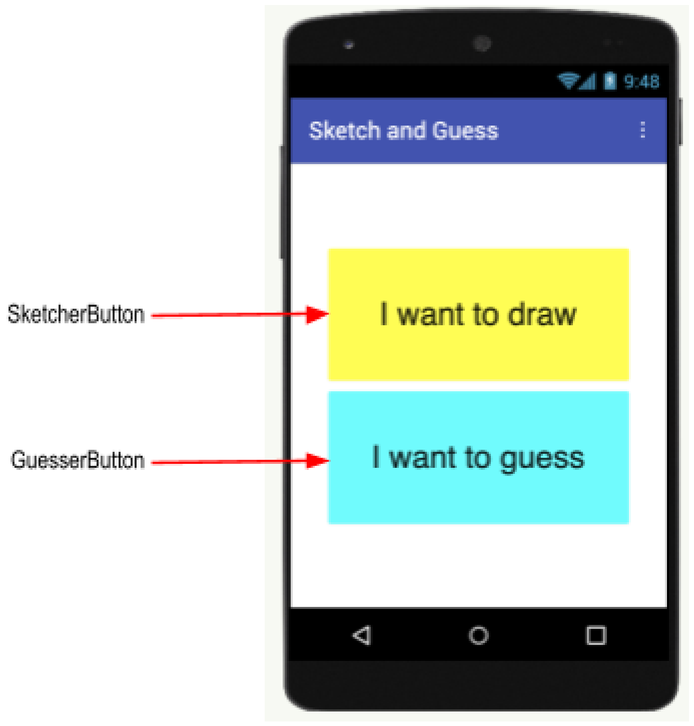
1. Create a new screen in App Inventor.
2. Navigate between screens in an App Inventor app.

Lesson Outline

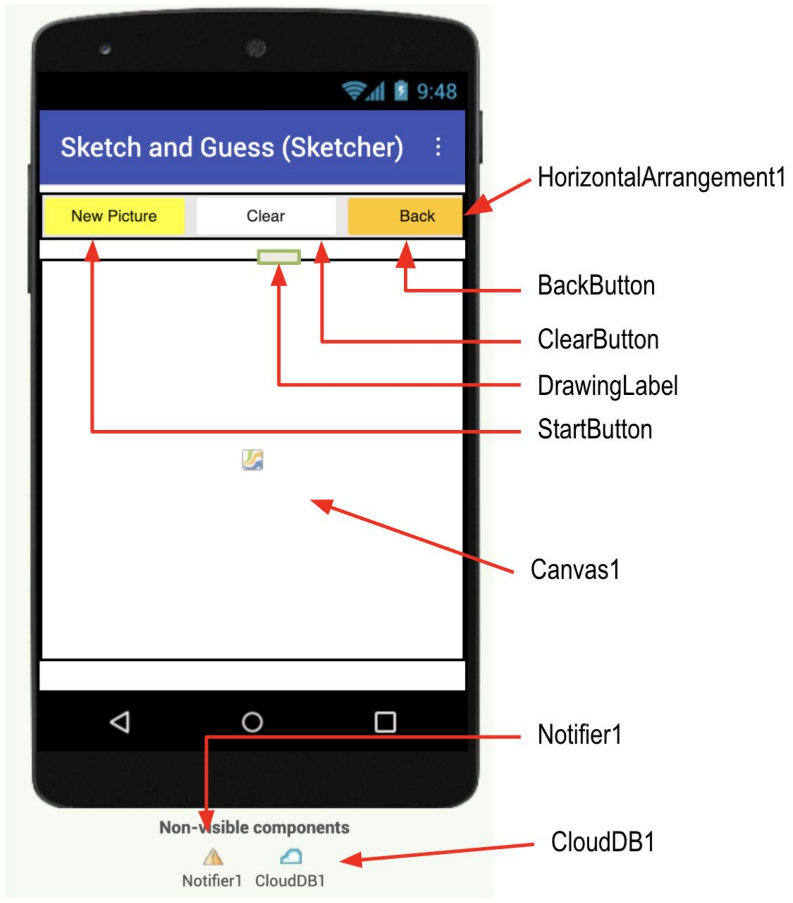
Introduction to App, Adding Multiple Screens (10 minutes)

1. Demonstrate a simple app with 2 screens. (SketchandGuess_checkpoint1.aia). This app will have the main screen, Screen1, that has two buttons. One button takes the user to the SketcherScreen, where they can draw. The other button is not functional at this point, but eventually will take the user to the GuesserScreen, where they can watch the Sketcher drawing, and guess what is being drawn.

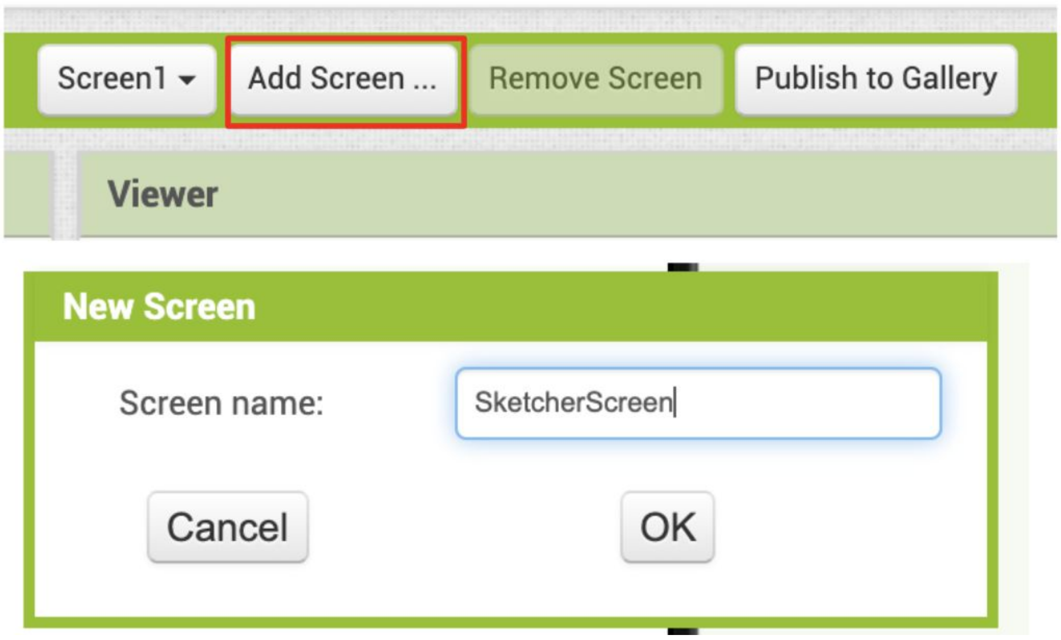
Screen1



SketcherScreen



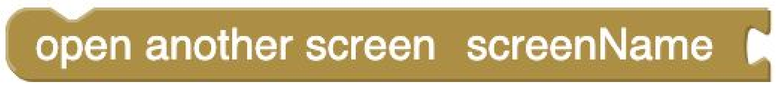
2. Explain that students will add multiple screens for this app and the drawing will happen on the SketcherScreen. The guessing will happen on the GuesserScreen.
3. In this lesson, students will build the user interface for the first two screens, and use the **open another screen** and **close screen** blocks to navigate between the screens.
4. Show students how to add new screens. First, click on the **Add Screen** button along the top of the IDE. Then, type in a name (no spaces) and click OK.

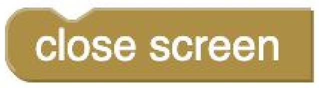


Screen name: SketcherScreen

Building Screens, Navigating Between Screens (30 minutes)

1. Show students the **open another screen** and **close screen** blocks in the Control panel. The **open another screen** block takes a text string as input. The screenName should match exactly the name of the screen to be opened.





The **close screen** block closes a screen. Students should understand that when a screen is opened in App Inventor, the new screen appears *on top of* the other screen, which is hidden behind the opened screen. So, when the user wishes to return to the original screen, the correct block to use is **close screen**, which removes the screen on top. The original screen is then uncovered. If you use **open another screen** to return to a screen, multiple layers of screens will be open in the app, eventually using up the app's memory.

2. Ask students to pair up and work using the Pair Programming model to build part 1 of Sketch and Guess. They will build Screen1 and the SketcherScreen with the help of *Student Guide: Lesson 1*, and navigate between the two screens with the **open another screen** and **close screen**.
3. Test and debug the app. Students should make sure they can navigate between Screen1 and SketcherScreen.

Wrap-up (5 minutes)

1. Review how to add a screen and how to navigate between the screens.
2. Explain that students will add code to draw on the SketcherScreen in Lesson 2.

Appendix 2
Unit 9 Teacher's Guide: Lesson 2

Learning Objectives

At the end of this lesson, students should be able to:

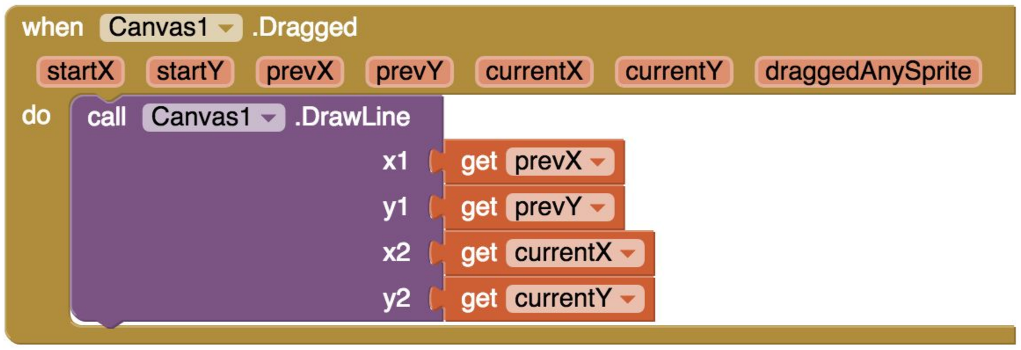
1. Draw using the Canvas component in App Inventor.

Lesson Outline

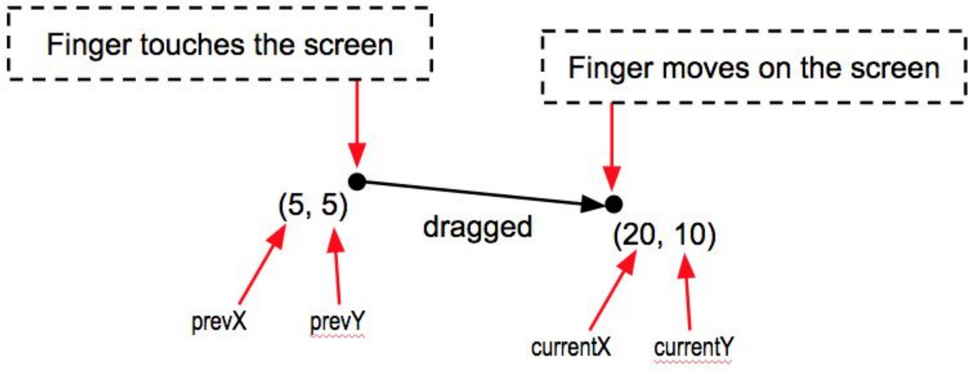
Introduction to Drawing with Canvas Component (10 minutes)

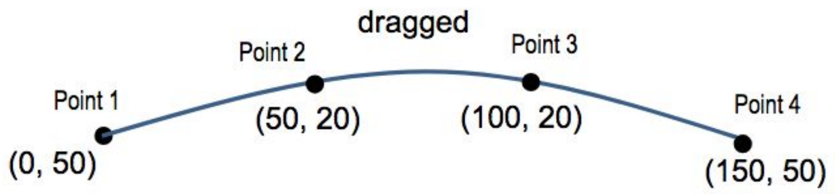
1. Demonstrate Sketch and Guess with the added drawing feature on the SketcherScreen (SketchandGuess_checkpoint2.aia).
2. Introduce drawing on a Canvas and how to draw a line on the Canvas.
    a. Show the Drawing and Animation tab in the Designer.
    b. Show a Canvas component added in the Designer
    c. Demonstrate the **Canvas.Dragged** event block and **Canvas.DrawLine** blocks.
        i. When the user drags a finger in the app, you want a line to be drawn where the finger is dragged. So, when the **Canvas.Dragged** event is triggered, you want to call **Canvas.DrawLine** to draw a line based on the start and end points of the drag. As the finger is dragged, the **Canvas.Dragged** event will be called thousands of times, so the line drawn will be thousands of very small lines connected together to form a

smooth curve.


```
when Canvas1 .Dragged
  startX startY prevX prevY currentX currentY draggedAnySprite
do call Canvas1 .DrawLine
        x1  get prevX
        y1  get prevY
        x2  get currentX
        y2  get currentY
```

d. Explain how a series of lines create a single line.





3. Ask students to fill out answers on pages 1 and 2 in *Student Guide: Part 2* to demonstrate understanding of these concepts.

Coding of Drawing Feature (30 minutes)

1. Ask students to pair up and work using the Pair Programming model to build a drawing app with the help of *Student Guide: Lesson 2.* The result should be a simple drawing app that allows the user to draw with a black line on the screen.
    a. Note that the drawing will take place on the SketcherScreen, so students should always return to Screen1 before testing. They will have to press the "I want to draw" button to open the SketcherScreen, where they can draw.
    b. The object that the user draws will be randomly chosen from a list of objects. The object will be displayed so the Sketcher knows what to draw.
2. Make sure students test and debug the app using the MIT AI2 Companion.

Wrap-up (5 minutes)

1. Review Canvas and drawing features learned in this lesson.
2. Ask students:
    a. "Have you played any drawing games before?" (Pictionary)
    b. "How can you make this app work like Pictionary?"
        - Hopefully students will relate the use of CloudDB to pass information between devices.

Appendix 3
Unit 9 Teacher's Guide: Lesson 3

Learning Objectives

At the end of this lesson, students should be able to:

1. Use CloudDB to pass drawing data between mobile devices.
2. Test an App Inventor app through the Build menu, to produce an apk that is installed on a mobile device.
3. Work collaboratively to build and test a multiplayer game.

Lesson Outline

Review of Drawing Feature (5 minutes)

1. Review the drawing feature added in Lesson 2.
2. Ask students how they can play this game with another person on a single device.
3. Ask students, "How can you play this game with partners who are using different devices?" (Hopefully students will suggest using CloudDB as a way to pass information between devices).
4. Teacher demonstrates the CloudDB version of the drawing app (SketchandGuess_checkpoint3.aia) to show Sketch and Guess between two devices.

Demonstration of Drawing Using CloudDB (10 minutes)

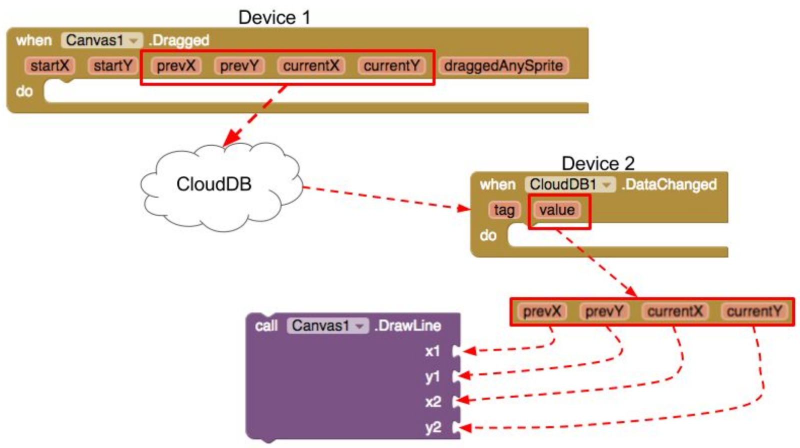
This unplugged activity will help students to strengthen their understanding of how CloudDB works, and how it can be used to send drawing information to other users.

Use the *CloudDB Unplugged Activity Storing Drawing Data* document to run the activity. One student, the Sketcher, will draw a polygon shape on a grid. With each line they draw, they will write the coordinates of the starting and ending point of the line segment, and pass it to a messenger who will deliver it to CloudDB. CloudDB will then hand it back and ask them to deliver it to the Guesser. The Guesser will take the numbers, and use the information to draw the same line segment on their paper. In the end, they should both have the same figure on their papers.

While this is not exactly how this will work in the app, where thousands of tiny line segments are drawn to create the drawing, this activity should simulate the general idea of passing line segment information via CloudDB.

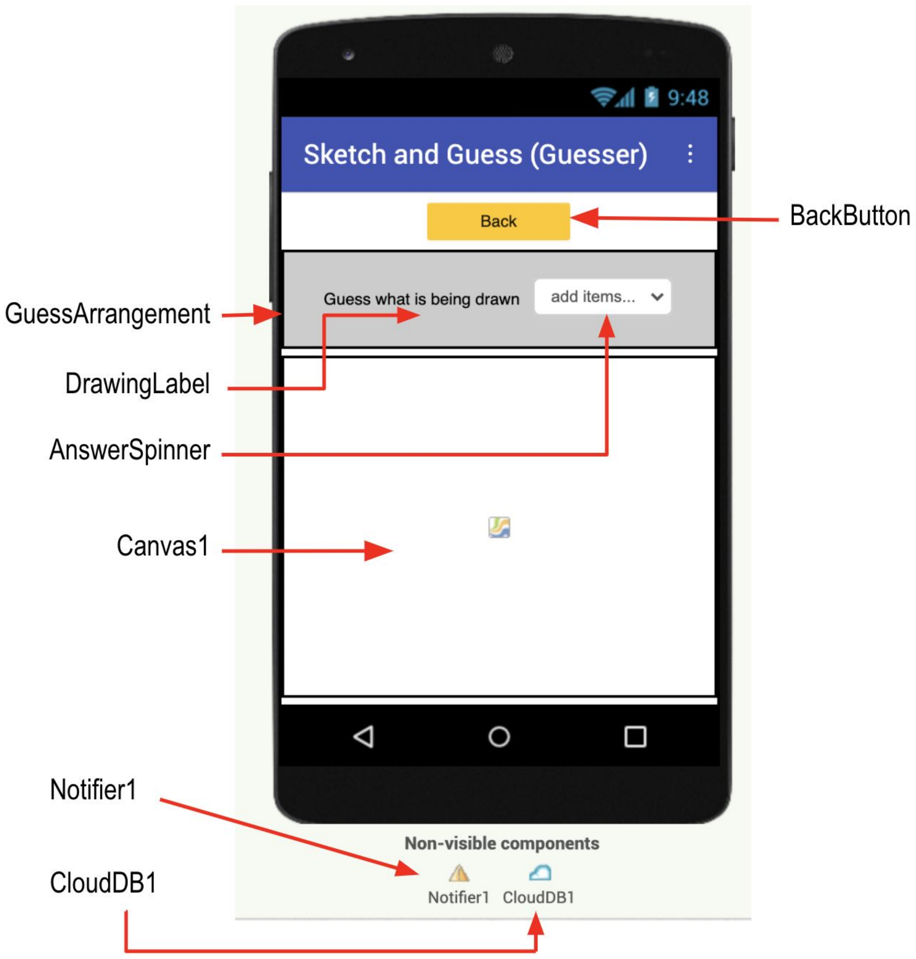
Adding CloudDB Component to the App (25 minutes)

1. Explain that whenever a user draws something on their device, they will also store the x,y coordinates for the start and end of the drag event to CloudDB. Other users will take that information and use it to draw the same line on their device.


Device 1
when Canvas1 .Dragged
startX startY prevX prevY currentX currentY draggedAnySprite
do
CloudDB
Device 2
when CloudDB1 .DataChanged
tag value
do
prevX prevY currentX currentY
call Canvas1 .DrawLine
x1
y1
x2
y2

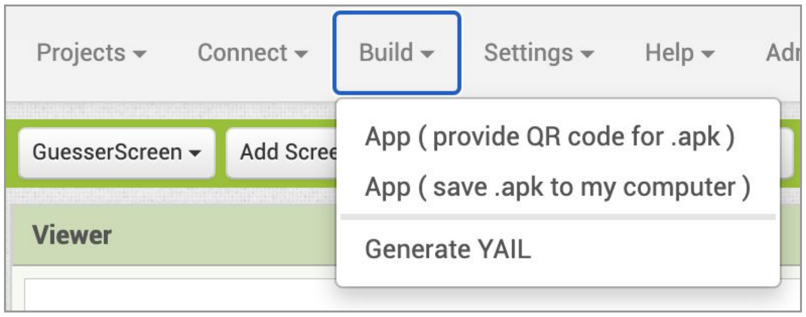
   Because there are 4 numbers (prevx,prevy,currentx,currenty) to be sent, the data will be put in a list, to be saved under a single tag

2. Ask students to work using the Pair Programming model, following *Student Guide: Lesson 3,* to add the third screen, GuesserScreen, to their apps.



They will also add CloudDB to their apps. The goal for this lesson is that if one person draws something on their app, it will appear on the other person's device.

3. Explain to students that in order to test their app, they will have to download an apk to each partner's device.
   a. Show students how to create an apk with the QR code option. From the Build menu, select "App (Provide QR code for apk)".



c. When the QR code appears (it might take a few minutes), both partners install the apk on their devices.
d. Students should test the app. One student will draw and the other can see what is being drawn on their device. Students will have to agree before testing who will be the Sketcher and who will be the Guesser, and press the corresponding button when they test their apps.

Wrap-up (5 minutes)

Ask students how they might make this into a game, where one person draws and other players guess. What is needed in the app to make the current app into a game? Students might suggest that the Guesser must have a way to make a guess, and let other players know if the answer is correct. In Part 4, there is one solution for making this into a game, much like Pictionary.

Appendix 4
Unit 9 Teacher's Guide: Lesson 4

Learning Objectives

At the end of this lesson, students should be able to:

1. Use the Spinner component to allow for app users to make a choice.
2. Use CloudDB to send game information between devices in order to make a working Sketch and Guess game app.

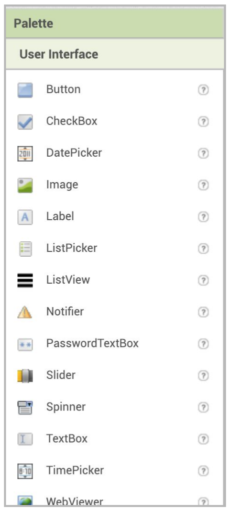
Lesson Outline

Review and Introduction to Lesson (10 minutes)

1. Review where the app stands now. Users can draw and others can see what they are drawing on their devices. Ask the whole class what else can be improved, and how they might make this into a Sketch and Guess game, where one person draws and other players guess.
2. The teacher demonstrates and explains to students that they will implement guess checking in this lesson by adding code to provide feedback to users when they guess. This should be familiar to students, as they have used if statements and the Notifier component in previous game apps to notify the user whether they have won or not.(SketchandGuess_checkpoint4.aia)
3. Show students how to add a Spinner component, and demonstrate how it works.

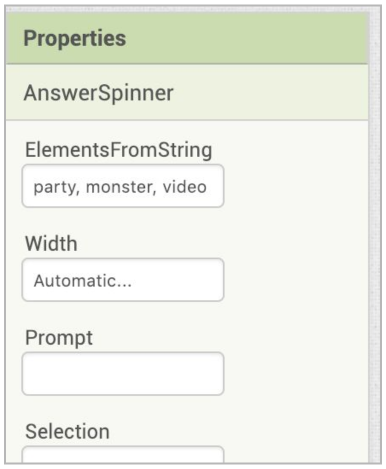
It appears in the User Interface drawer.

It is similar to ListView and ListPicker components



You can set the elements of the Spinner in the

Properties panel in the Designer.



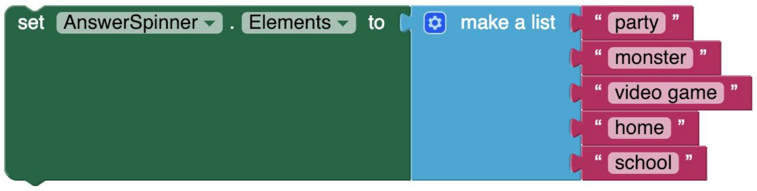
ElementsFromString: party, monster, video

Width: Automatic...

Prompt:

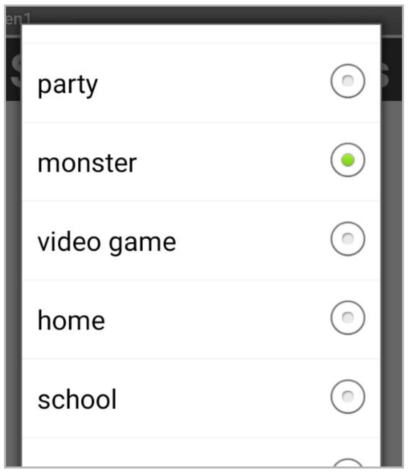
Selection:

Or you can set it programmatically.

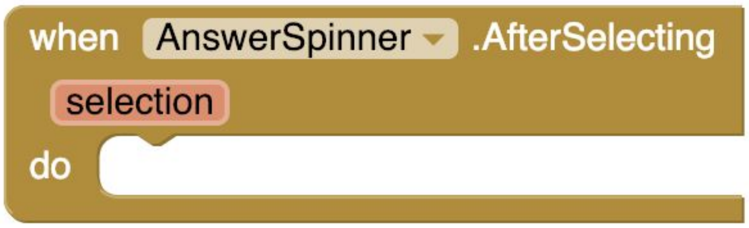


```
set AnswerSpinner . Elements to make a list "party"
                                            "monster"
                                            "video game"
                                            "home"
                                            "school"
```

It appears as a list with radio buttons as choices.



After the user makes their choice, there is an **AfterChoosing** event. App inventors can test for which selection the user chose. The **selection** parameter contains what the user has selected.



Coding the Sketch and Guess App (20 minutes)

Following *Student Guide: Lesson 4*, students add code to their apps to make a fully working "Sketch and Guess" game. Students will add code so Guessers can select from the Spinner dropdown list which object they think is being drawn. The app will check for a correct answer, and notify the Guesser if they are correct or incorrect with their guess. All users will be notified, using CloudDB, if the Guesser was correct. The students can choose to switch roles, by returning to Screen1, and pressing the alternate button to their previous role.

Any student pairs who complete Lesson 4 coding may attempt the *Student Guide: Challenge* where they can add color buttons and a slider to change the line width in the app.

Wrap-up (5 minutes)

1. Check in with students to see where they are with the app. Consider how much time will be needed by student groups to complete the app.
2. Explain that students have one more lesson to complete the app, try the Challenge, or add a different feature themselves.

Appendix 5
Unit 9 Teacher's Guide: Lesson 5

Learning Objectives

At the end of this lesson, students should be able to

1. Collaboratively test and debug an app until it works correctly and as expected.
2. Add a new feature to the standard app, such as color, line width, or images.

Lesson Outline

Introduction to Lesson (5 minutes)

1. Check in again with student groups to see where they are with the app..
2. Students can either finish the app if they have not completed it, they can try the *Student Guide: Challenge*, or they can add their own new feature. There are some suggestions at the end of the Challenge.

Coding (25 minutes)

1. Any students still working on Parts 1-4 may continue working to complete the standard app.
2. Students can choose to try the Challenge, adding color buttons and a slider to change the paintbrush size.
3. Students can also choose to extend the app by adding their own features.

Wrap-up (15 minutes)

1. Review the use of CloudDB in this unit.
2. Ask students to reflect on the app, and ask for volunteers to share any new features added.
3. Ask students to answer multiple choice questions and learning attitudes survey.