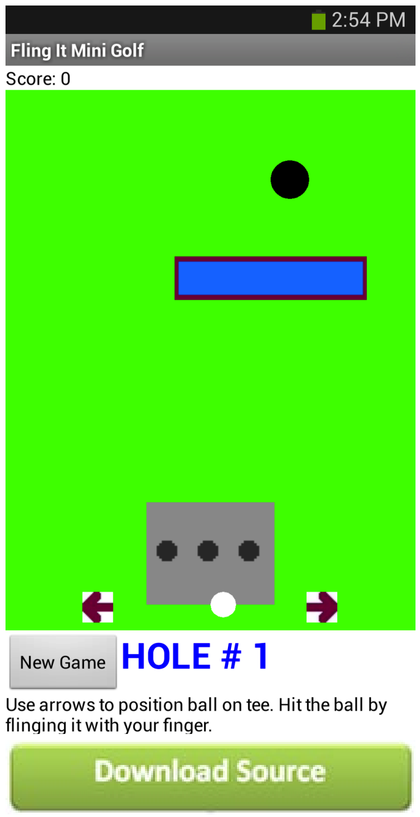
MIT APP INVENTOR

Create Apps!

This Mini Golf App demonstrates how to use the Fling, TouchUp, and TouchDown gestures for Sprites Note that these event handlers are also available for the Canvas.


2:54 PM
Fling It Mini Golf
Score: 0
New Game
HOLE # 1
Use arrows to position ball on tee. Hit the ball by flinging it with your finger.
Download Source

To play this mini golf app, the player first positions his/her ball within the confines of the tee, and then flings the ball toward the hole. The ball will bounce off of the rectangular obstacle and the sides of the course. For each fling of the ball, the stroke count goes up by one. The total score is the number of strokes it takes to complete the entire course.

This tutorial covers:

- Using the **Sprite** component and the TouchUp, TouchDown, and Flung events
- Using a **Clock** component
- Dynamic Positioning of sprites on a canvas, based on the size of the screen
- Sprite Collisions

This tutorial assumes you are familiar with the basics of App Inventor--using the Component Designer to build a user interface, and using the Blocks Editor to specify the app's behavior. If you are not familiar with the basics, try stepping through some of the basic tutorials before continuing.

Part I: Start a new app and make a ball that responds to fling events

We'll build this app in stages, adding a little bit of the game at a time. Log into App Inventor and start a new project. Name it "MiniGolf". When the Design window opens notice that App Inventor automatically names the screen "Screen1", but you can set the Title of the screen, which will show up in the top bar of the app. Think of a title related to Mini Golf, or feel free to use the suggested title "Fling It Mini Golf", and type it into the Properties pane on the right side of the Designer.

In the Screen Properties (shown in right-hand pane): Uncheck the checkbox labeled "Scrollable" so that the screen will not scroll when the app is running. Screens that are set to scroll do not have a height. We'll need our screen to have a defined height in order to set up the golf course properly.

Add the following components in the Designer:

Component Type	Palette Group	What You'll Name It	Purpose	Properties
Canvas	Drawing and Animation	Canvas1	The canvas serves as the golf course	Height: 300 Width: FillParent BackgroundColor: Green (or whatever you like!)
Ball	Drawing and Animation	GolfBall	This is the ball the player will fling to try to hit the Hole	Radius = 10 Color: White (or your choice!) Speed: 0 Interval: 1 (ms) Z = 2 (when sprites are overlapping, the one with the higher z will appear on top)
Ball	Drawing and Animation	Hole	This will be the target for the GolfBall	Radius = 15 Color: Black Speed: 0
Clock	Sensors	Clock1	The clock will fire continuously to control the movement of the ball	Timer Always Fires Timer Enabled TimerInterval: 100

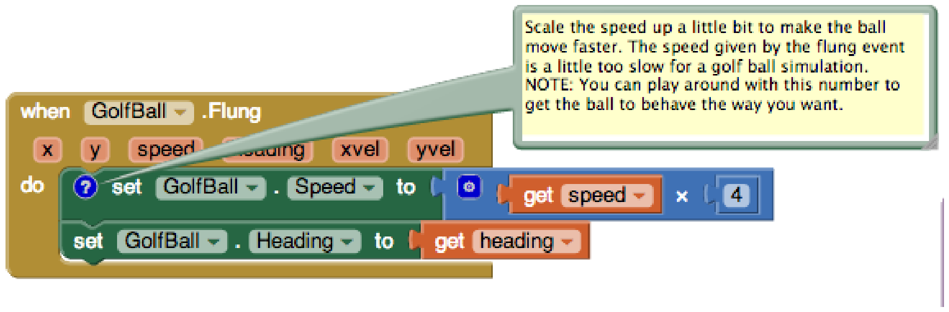
Open the Blocks Editor

Program the behavior of the Ball:

First, use the GolfBall.Flung event handler to move the golf ball when it is flung. Notice how this event handler takes in 6 different arguments:

- ***x*, the x position on the Canvas grid of the user's finger**
- ***y*, the y position on the Canvas grid of the user's finger**
- ***speed*, the speed of the user's flinging gesture**
- ***heading*, the direction (in degrees) of the user's fling gesture**
- ***xvel*, the speed in the x direction of the user's fling**
- ***yvel*, the speed in the y direction of the user's fling**

Essentially, you want to set the GolfBall's speed and heading to match the speed and heading of the player's fling gesture. You may want to scale up the speed a little bit because the speed of the fling is a little slower than how a golf ball would move. You can play with this "scaling factor" to make the ball more or less responsive to a fling.



```
when GolfBall .Flung
  x  y  speed  heading  xvel  yvel
do  set GolfBall . Speed to  get speed × 4
    set GolfBall . Heading to  get heading
```

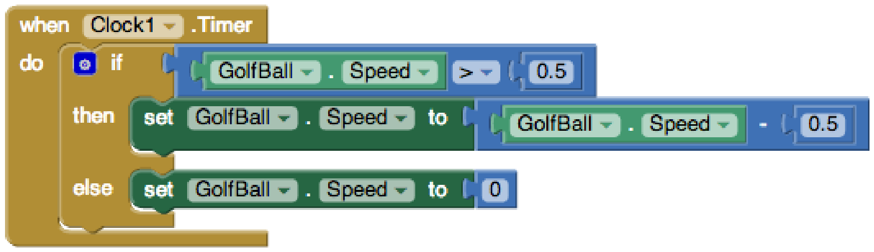
Scale the speed up a little bit to make the ball move faster. The speed given by the flung event is a little too slow for a golf ball simulation.
NOTE: You can play around with this number to get the ball to behave the way you want.

Program the behavior of the clock:

Use timer event to slow ball down so it doesn't bounce around forever.

Each time the clock fires, it will reduce the speed of the ball slightly. Notice that if the ball is not moving then these blocks will do nothing. If you don't have this then the ball will just bounce forever.

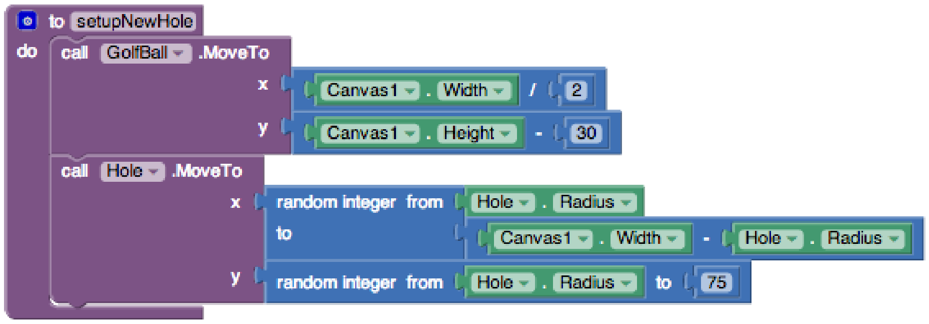
You'll need to use the if mutator function to change the if block into an if-else block. For a summary of mutators, check out the Mutators page



```
when Clock1 .Timer
do  if    GolfBall . Speed > 0.5
    then  set GolfBall . Speed to  GolfBall . Speed - 0.5
    else  set GolfBall . Speed to 0
```

Program a new procedure called SetupNewHole:

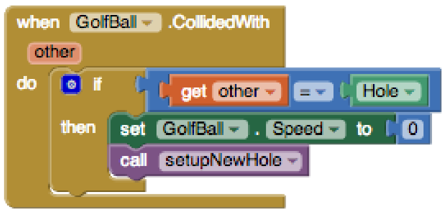
This procedure will be called when a hole is scored and the ball has to be placed back at the starting point. Note that the Hole.MoveTo block sets the hole up in a new random location for the next play.



```
to setupNewHole
do  call GolfBall .MoveTo
         x  Canvas1 . Width / 2
         y  Canvas1 . Height - 30
    call Hole .MoveTo
         x  random integer from  Hole . Radius
                           to    Canvas1 . Width - Hole . Radius
         y  random integer from  Hole . Radius  to 75
```

Program the Behavior of the Hole: When the ball collides with the hole, the ball disappears and resets at the bottom of the screen.

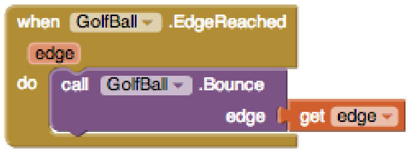
Note: When you first drag out the GolfBall.CollidedWith event handler, the named parameter is called "other". Notice that the if then block tests to see if the object involved in the collision with the golf ball (other) is the black ball sprite representing the hole. You can't just put a text block with the word "Hole" in it, you must use the Hole block, that can be found in the drawer for the Hole image sprite. Do not use a text block here.



```
Test this Behavior. Connect your device to AppInventor, or start the emulator to load your app. When
you fling the ball it should move in the direction of your fling, with a speed similar to the
strength of your fling. The ball should slow down as it moves, eventually stopping. When the ball
hits the hole, the ball should reset at the bottom of the screen and the hole should move to a new
random location.
```

Does your ball get stuck if it hits the edge?

This is easy to fix with the when EdgeReached event. Note that you can find the "edge" value block by using a get block and selecting "edge" from the dropdown.



Double check to make sure your code is right: fling the ball a few times and see that that ball now bounces off the edges of the course.

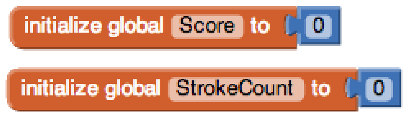
Part II: Keeping Score

Games are more fun if you have a way to see how you're doing. Let's add a stroke counter. In mini golf your score goes up as you take more strokes. The goal is to have the lowest score possible. Let's show the player how many strokes she or he has taken on this hole. Let's also show the number of strokes taken during the whole game.

Go back to the Designer and set up the following components:

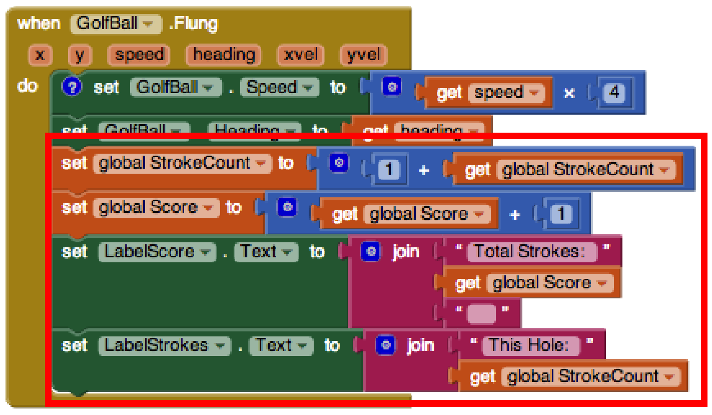
Component Type	Palette Group	What You'll Name It	Purpose	Properties
Horizontal Arrangement	Layout	HorizontalArrangement1	Contains LabelScore and LabelStroke	Place at top of screen
Label	User Interface	LabelScore	Displays the total stroke count for the entire game	
Label	User Interface	LabelStroke	Displays the stroke count for the hole the player is currently on	

In the Blocks Editor, you can program updates to the Score and Stroke labels. First, set two new global variables called StrokeCount and Score, and set their initial values to 0.



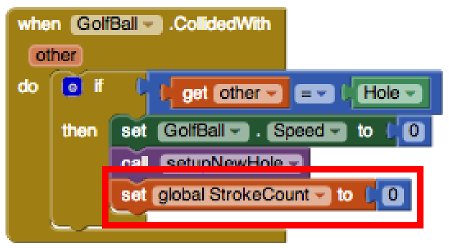
```
initialize global Score to 0
initialize global StrokeCount to 0
```

Then add the following blocks to the GolfBall.Flung event (red rectangle indicates new blocks):



```
when GolfBall.Flung
  x  y  speed  heading  xvel  yvel
do  set GolfBall.Speed to  get speed × 4
    set GolfBall.Heading to  get heading
    set global StrokeCount to  1 + get global StrokeCount
    set global Score to  get global Score + 1
    set LabelScore.Text to  join  " Total Strokes: "
                                  get global Score
                                  " "
    set LabelStrokes.Text to  join  " This Hole: "
                                    get global StrokeCount
```

Next add the following blocks to the Event that handles the ball hitting the hole:



```
when GolfBall.CollidedWith
  other
do  if  get other = Hole
    then  set GolfBall.Speed to 0
          call setupNewHole
          set global StrokeCount to 0
```

```
Test the behavior. With these new changes, you should have a "Total Strokes" count and "This Hole"
count at the top of the screen. When you fling the ball, the "This Hole" count  and "Total Strokes"
count should both increase by one, and when you make the ball go into the hole the "This Hole" count
should reset to 0.
```

Part III: Positioning Ball on Tee using TouchUp and TouchDown events

Ok, so now you've got a working game! Now let's make it a little more interesting and fun. First we'll add a Tee and let the player position the golf ball on the tee before they fling the ball.

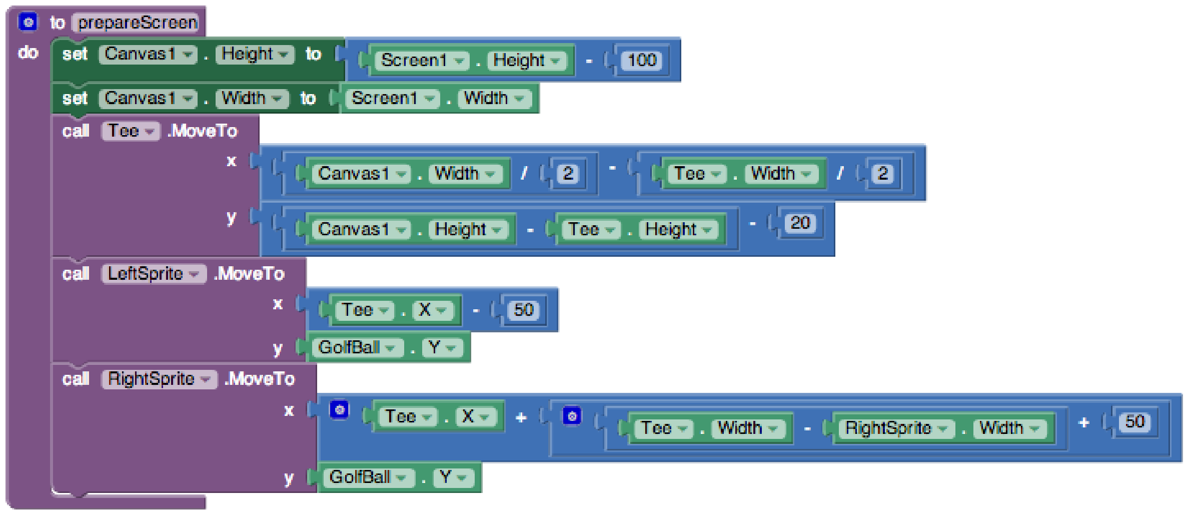
Go back to the Designer and add three new image sprite components:

Component Type	Palette Group	What You'll Name It	Purpose	Properties
ImageSprite	**Drawing and Animation**	**Tee**	**A rectangular area in which the player can position their ball before teeing off.**	**Upload the Tee image (right click on this link, or see below).**
ImageSprite	**Drawing and Animation**	**LeftSprite**	**This is a left pointing arrow that the player will use to move the ball to the left on the tee**	**Upload the left arrow graphic (right click on this link**

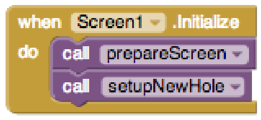
Drawing and Animation	ImageSprite	RightSprite	This is a right pointing arrow that the player will use to move the ball to the left on the tee	Upload the right arrow graphic (right click on this link

Program the size of the canvas, and the placement of the ball and image sprites on the canvas:

First, program the setup of these components on the screen. It's best to accommodate all different screen sizes by placing the sprites on the screen relative to the size of the screen. The blocks below show how to set up the screen *dynamically* so that everything fits the right way. We start off by making the canvas size based on the screen size, and then we place each sprite in relation to the width and height of the canvas. We'll make a procedure to do this for us. Try to understand all of these blocks before you move on.



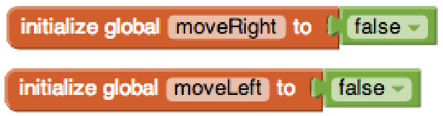
```
to prepareScreen
do  set Canvas1.Height to ( Screen1.Height - 100 )
    set Canvas1.Width to Screen1.Width
    call Tee.MoveTo
        x ( ( Canvas1.Width / 2 ) - ( Tee.Width / 2 ) )
        y ( ( Canvas1.Height - Tee.Height ) - 20 )
    call LeftSprite.MoveTo
        x ( Tee.X - 50 )
        y GolfBall.Y
    call RightSprite.MoveTo
        x ( Tee.X + ( ( Tee.Width - RightSprite.Width ) + 50 ) )
        y GolfBall.Y
```



```
when Screen1.Initialize
do  call prepareScreen
    call setupNewHole
```

Position the Golf Ball on the Tee using TouchUp and TouchDown on the Arrow sprites:

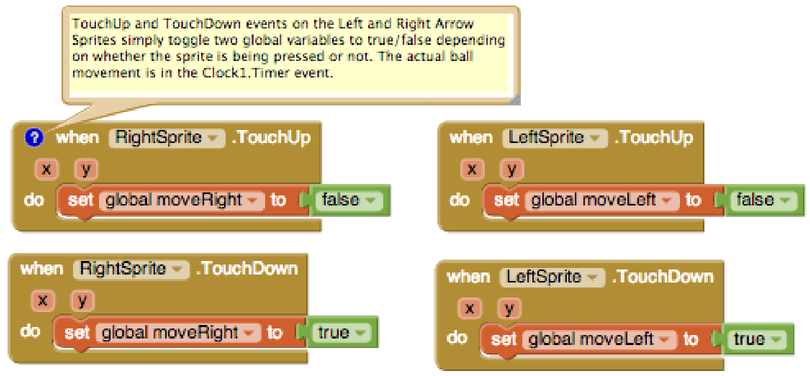
To handle this, first set up two global variables that are toggled each time an arrow is pressed.



```
initialize global moveRight to false
initialize global moveLeft to false
```

Program the behavior of the Right and Left Arrows

The left and right arrows are image sprites, so they come equipped with the ability to know when the player is is holding his/her finger down on them. The following blocks toggle the global variables based on whether the user is pressing either of these arrows.



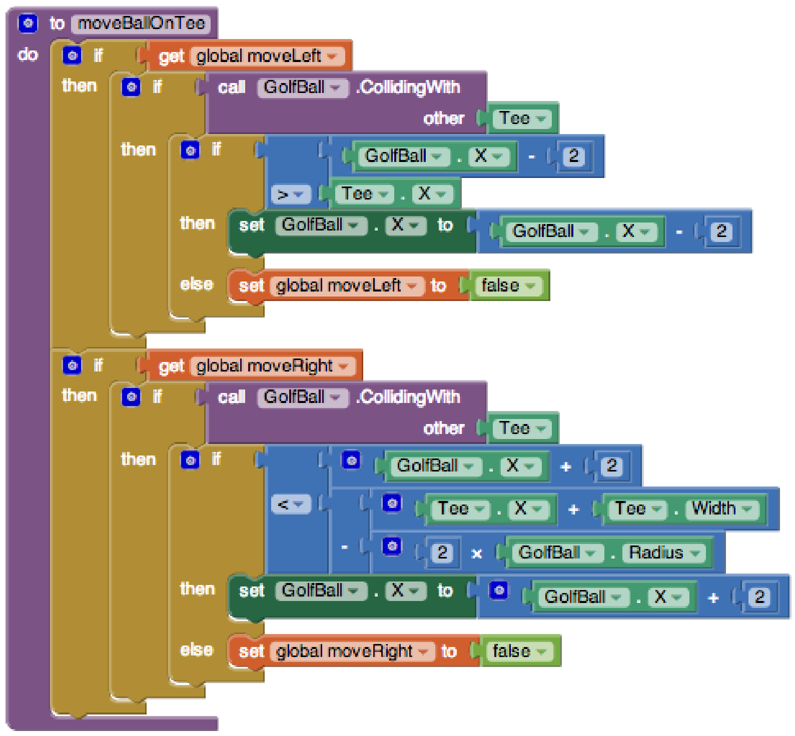
TouchUp and TouchDown events on the Left and Right Arrow Sprites simply toggle two global variables to true/false depending on whether the sprite is being pressed or not. The actual ball movement is in the Clock1.Timer event.

```
when RightSprite .TouchUp
  x  y
do  set global moveRight to  false

when LeftSprite .TouchUp
  x  y
do  set global moveLeft to  false

when RightSprite .TouchDown
  x  y
do  set global moveRight to  true

when LeftSprite .TouchDown
  x  y
do  set global moveLeft to  true
```

Procedure MoveBallOnTee:

Make a new procedure moveBallOnTee that makes the golf ball move left or right on the tee depending on the global variables. Although the math here looks complicated, it’s pretty simple. If the ball is supposed to move left, you first check to make sure that moving the ball 2 pixels left will not exceed the left-most coordinate of the Tee. If moving the golf ball to the right, you first check that moving the ball right 2 pixels will not move it past the right-most coordinate of the Tee.

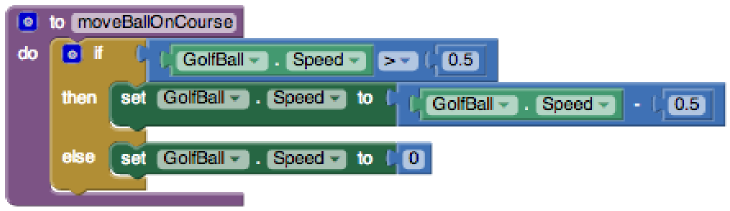
Note: if blocks look different in this image than on your own screen, this is because they were aligned differently. If you right click on the blocks, a list of options will appear and one of them is external inputs. When you select this, it will change how the blocks are configured. It does not change how the blocks function. If you want to change this, right click again and select internal inputs.



```
to moveBallOnTee
do  if  get global moveLeft
    then  if  call GolfBall .CollidingWith
                          other  Tee
          then  if  GolfBall . X  -  2
                    >
                    Tee . X
                then  set GolfBall . X to  GolfBall . X  -  2
                else  set global moveLeft to  false
    if  get global moveRight
    then  if  call GolfBall .CollidingWith
                          other  Tee
          then  if  GolfBall . X  +  2
                    <
                    Tee . X  +  Tee . Width
                    -  2  ×  GolfBall . Radius
                then  set GolfBall . X to  GolfBall . X  +  2
                else  set global moveRight to  false
```

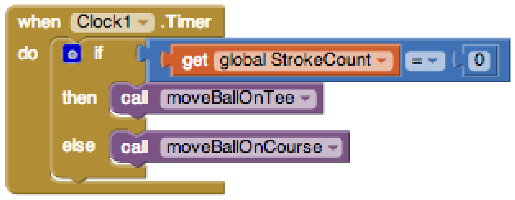
MoveBallOnCourse Procedure

Note that the blocks that we had inside the Clock1.Timer event are now moved over to a new procedure called moveBallOnCourse:



```
to moveBallOnCourse
do  if    GolfBall . Speed > 0.5
    then  set GolfBall . Speed to  GolfBall . Speed - 0.5
    else  set GolfBall . Speed to  0
```

On each new course, players can position the ball on the tee before attempting to fling the ball toward the hole. To program this, you first have to check to make sure this is a new course and the ball has not been flung yet. If StrokeCount = 0 then we know this course is brand new and the player has not yet attempted to get the ball into the hole.



```
when Clock1 .Timer
do  if    get global StrokeCount = 0
    then  call moveBallOnTee
    else  call moveBallOnCourse
```

As the blocks above show, after verifying that the StrokeCount is 0, you then want to proceed to move the golf ball left or right depending on which arrow is being pressed.

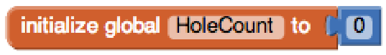
```
Test the behavior. Make sure your app is doing what you expect: play the game on your device or
emulator. Before you tee off, are you able to move the ball left and right on the tee by using the
left and right arrows? After you tee off, you should no longer be able to use the left and right
arrows (pressing them will do nothing). After the ball goes into the hole and the screen resets, you
should then be able to move the ball left and right on the tee before teeing off again.
```

Keep track of the number of holes played, and allow a game reset

The game is working pretty well now, but what about giving the player a way to reset the game? Also, it would be nice to give the player some instructions so they know how to play the game. While we’re at it, let’s also give an indication of how many holes the player has completed. Add the following components in the Designer:

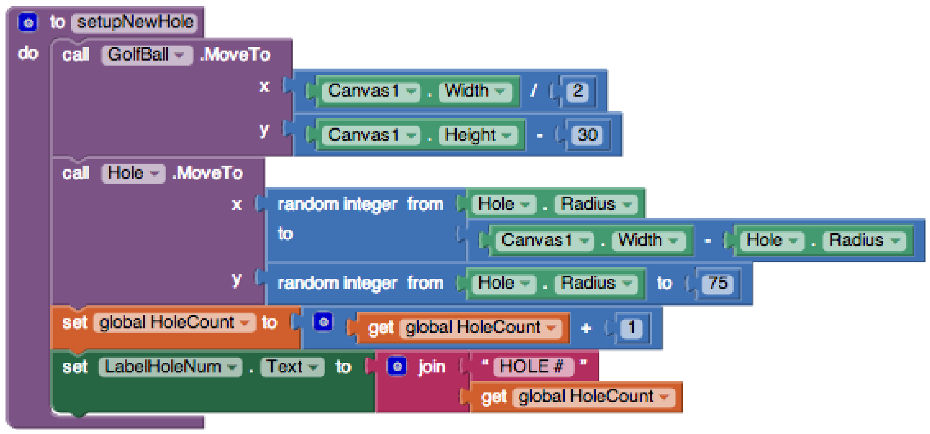
Component Type	Palette Group	What You’ll Name It	Purpose	Properties
Horizontal Arrangement	Layout	Horizontal Arrangement2	Contains the NewGame button and the HoleNum label	
Button	User Interface	ButtonNewGame	Resets the game to Hole #1 with a score of 0.	Text: "New Game"
Label	User Interface	LabelHoleNum	Displays the current hole number, increments by one each time a hole is completed.	Text = "Hole # 1" Font: bold, 28, blue
Label	User Interface	LabelInstruct	Displays instructions	Text = "Use arrows to position ball on tee. Hit the ball by flinging it with your finger."

Define a new global variable to keep track of the Hole Number:



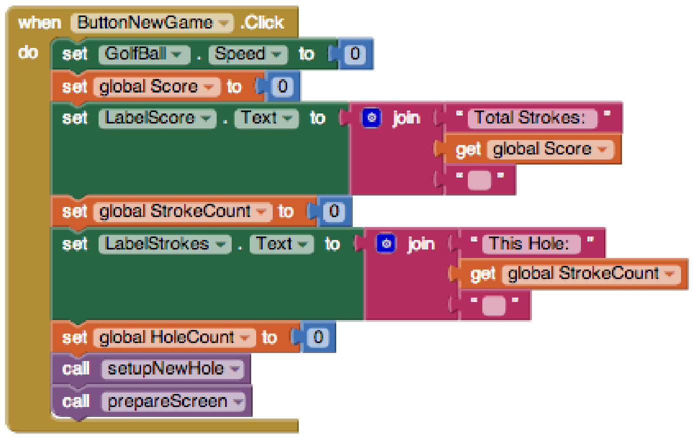
```
initialize global HoleCount to 0
```

Add the following blocks to the setupNewHole procedure: set global HoleCount and set LabelHoleNum.Text...



```
to setupNewHole
do  call GolfBall.MoveTo
        x  Canvas1.Width / 2
        y  Canvas1.Height - 30
    call Hole.MoveTo
        x  random integer from Hole.Radius
                          to   Canvas1.Width - Hole.Radius
        y  random integer from Hole.Radius to 75
    set global HoleCount to  get global HoleCount + 1
    set LabelHoleNum.Text to  join " HOLE # "
                                   get global HoleCount
```

Program the "New Game" button's behavior, which is pretty simple. When the button is pressed, set up a new course and reset both the hole stroke counter and total stroke counter to zero. Also set the hole number back to 1, by displaying "Hole #1" in LabelHoleNum. The blocks look like this:



```
when ButtonNewGame.Click
do  set GolfBall.Speed to 0
    set global Score to 0
    set LabelScore.Text to  join " Total Strokes: "
                                 get global Score
                                 " "
    set global StrokeCount to 0
    set LabelStrokes.Text to  join " This Hole: "
                                   get global StrokeCount
                                   " "
    set global HoleCount to 0
    call setupNewHole
    call prepareScreen
```

```
Test the behavior.
```

```
 Go back to your device or emulator and play the game some more. Now you should see the Hole #
displayed in the lower right. Hitting "New Game" button should reset the game, returning both scores
to 0, resetting the screen, and setting the Hole number to #1.
```

Part IV: Introduce an Obstacle

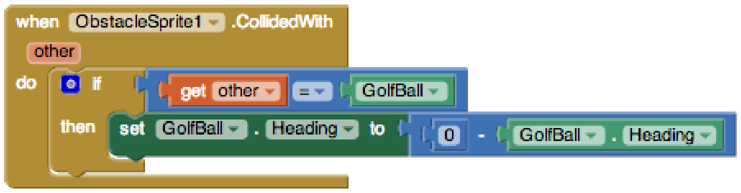
Most mini golf courses have obstacles on them. Let's add a simple rectangular obstacle that will randomly position itself on the course somewhere between the Tee and the Hole. Each time a new course is presented, the obstacle will move, just the same way the Hole moves each time a new course is set up.

Add the following component in the Designer:

Component Type	**Palette Group**	**What You'll Name It**	**Purpose**	**Properties**

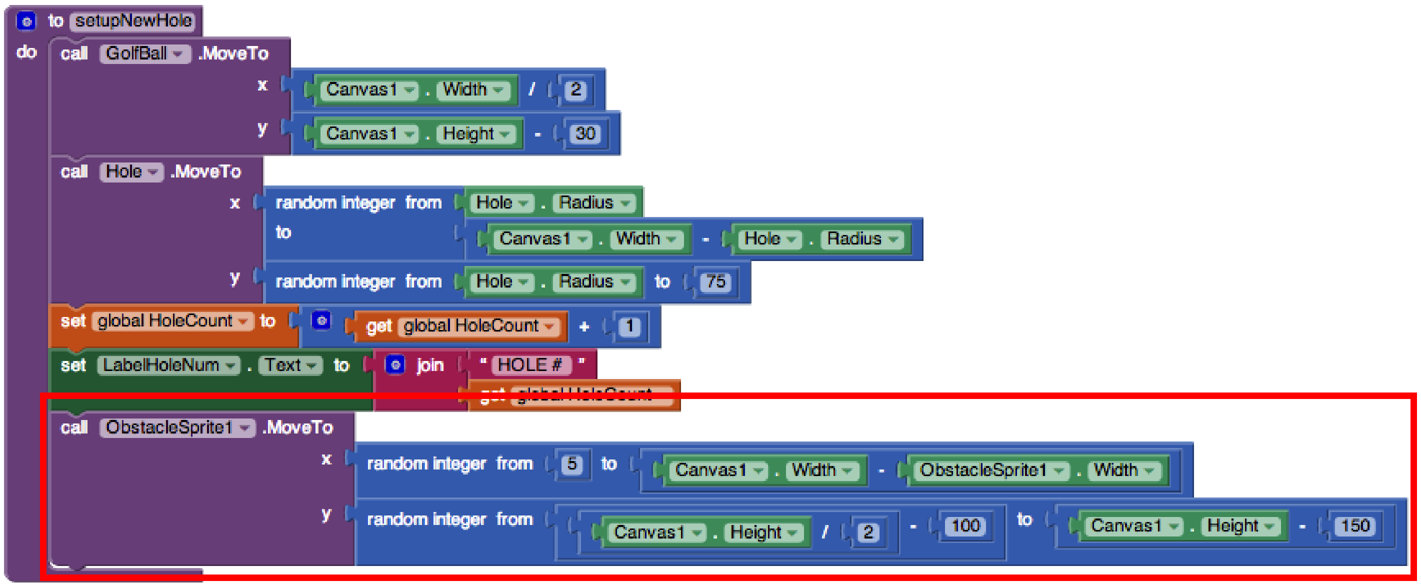
ImageSpriteand	Drawing Animation	ObstacleSprite1	This sprite will be somewhere between the golf ball and hole and will make it harder to get the ball into the hole	Upload the obstacle (rectangle) graphic (right click on this link, or see below).

Program the behavior of the obstacle in the blocks editor. First, set the behavior for when the ball hits the obstacle. *Note: Using Heading = 0 - heading works because we are dealing with bouncing off of horizonal surfaces, this will not work for bouncing off of vertical or inclined surfaces. For our purposes, it works all right. See Challenge #2 below for more information.



```
when ObstacleSprite1 .CollidedWith
  other
do  if   get other = GolfBall
    then set GolfBall . Heading to 0 - GolfBall . Heading
```

Each time the course is reset, position the obstacle will be positioned randomly. Add these blocks to the setupNewHole procedure:



```
to setupNewHole
do  call GolfBall .MoveTo
         x  Canvas1 . Width / 2
         y  Canvas1 . Height - 30
    call Hole .MoveTo
         x  random integer from Hole . Radius
                           to   Canvas1 . Width - Hole . Radius
         y  random integer from Hole . Radius to 75
    set global HoleCount to get global HoleCount + 1
    set LabelHoleNum . Text to join " HOLE # "
                                   get global HoleCount
    call ObstacleSprite1 .MoveTo
         x  random integer from 5 to Canvas1 . Width - ObstacleSprite1 . Width
         y  random integer from Canvas1 . Height / 2 - 100 to Canvas1 . Height - 150
```

```
Test the behavior. Play the game again. Notice that the ball bounces off when it hits the obstacle.
When the ball goes into the hole, or when the New Game button is pressed, the obstacle appears in a
new location somewhere between the tee and the hole.
```

That's it! Share your game with friends by building an APK or by downloading the source and sharing the zip file with other App Inventors!

Part V: Challenges

Here are some extra challenges to make your game better.

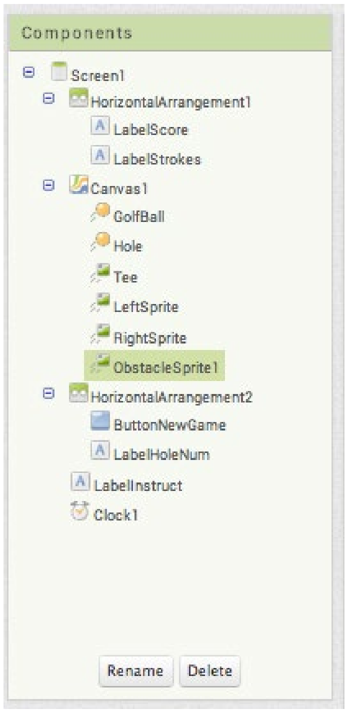
Challenge 1: Program the Ball to Hole collision so that the ball only goes into the hole if the golf ball's speed is not too fast. In real mini golf, the ball will bounce right over the hole if you hit the ball too hard.

Challenge 2: There is a slight bug when the ball hits the vertical sides of obstacle. Figure out how to program the ball to obstacle collision so that it bounces the way you would expect when the ball hits

the vertical sides.

Challenge 3: Limit the number of holes per game. Keep track of the number of holes and end the game after a set number. (A typical mini golf course has 18 holes.)

Below is a summary of all of the components:



Scan the following barcode onto your phone to install and run the sample app.

Download Source Code

If you'd like to work with this sample in App Inventor, download the source code to your computer, then open App Inventor, click Projects, choose Import project (.aia) from my computer..., and select the source code you just downloaded.

`Done with Mini Golf? Return to the other tutorials.`