|  |  |
| --- | --- |
| *Turn Off Lights* is a variation of the famous old [Whack-a-Mole](https://en.wikipedia.org/wiki/Whac-A-Mole) game -- this one promoting the socially useful message of saving electricity. In the game, a light bulb (represented by an *ImageSprite*) pops up at random positions on the screen. A player can score by touching the light bulb before it disappears.    This app uses animation, a clock, and randomness to move the ImageSprite around the canvas. This tutorial guides you through the basic steps in creating the animation.    **Objectives:** In this lesson you will learn to :   * follow an instructor-led walkthrough to create the *TurnOffLights*  app on a mobile device; * continue learning how to navigate the App Inventor online programming platform; * develop your understanding of what an App Inventor program is; * develop your understanding of how timing, animation, and randomness are programmed; * develop your understanding of procedures and procedural abstraction; * deepen your understanding of event-driven programming. | screenshotPart1.png  ***[Click the image to watch video](http://www.youtube.com/watch?v=t6rpDBelpqU)*** |

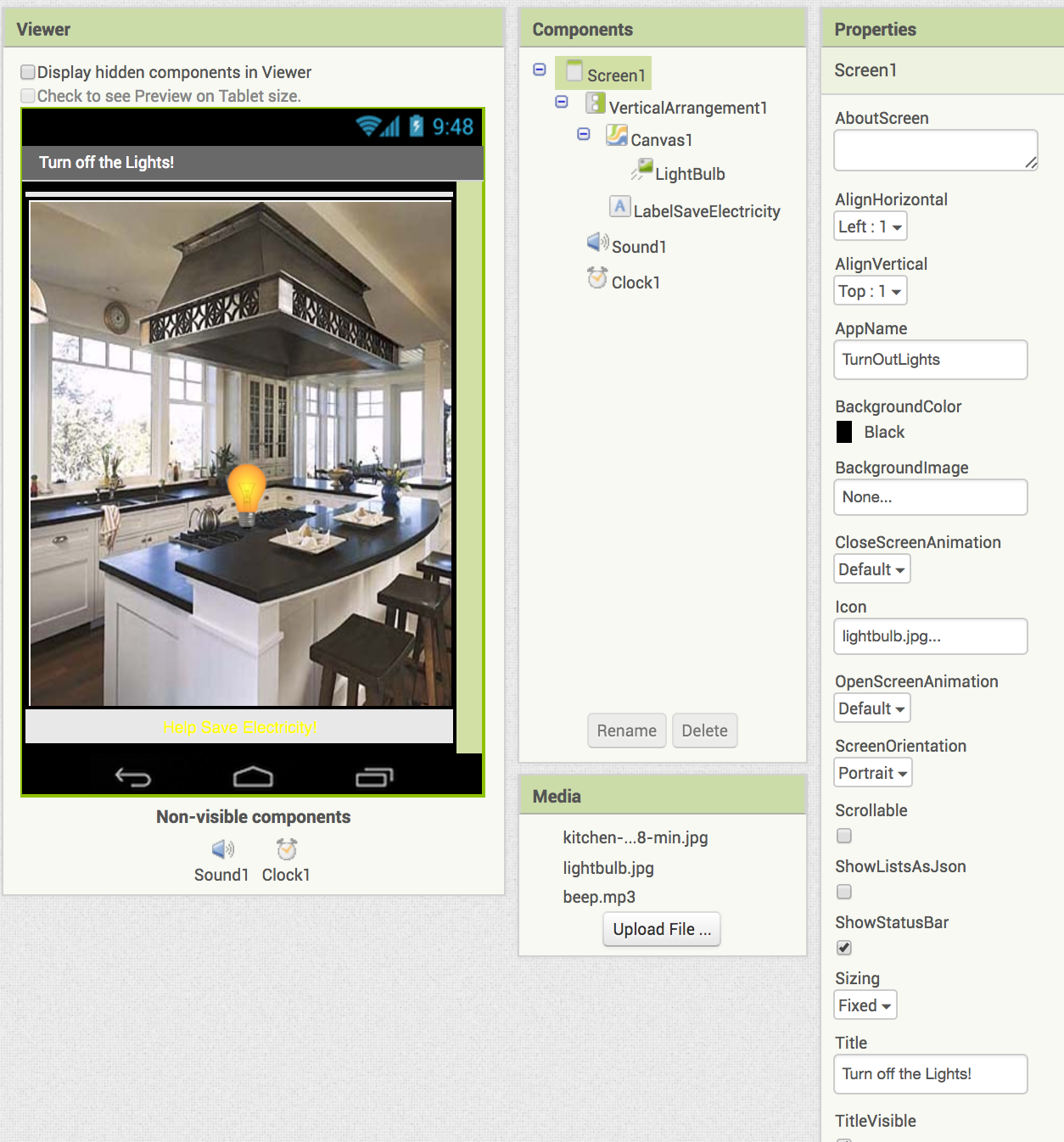
# Getting Ready

Open App Inventor with the [Turn Off Lights template](http://ai2.appinventor.mit.edu/?repo=templates.appinventor.mit.edu/trincoll/csp/unit4/templates/LightsOffTemplate/TurnOffLightsTemplate.asc). The template app just contains the images and sound files for the app, but no design or code yet.

## 

## 

## The Turn Off Lights User Interface (UI)



The UI for our this app will consist of four types of *Components:*

* *Basic Components*: Canvas, Button, Label, Clock
* *Media Component*: Sound
* *Animation Component*: an ImageSprite
* *Layout Component*: Vertical Arrangement

The *Canvas* serves as the background where the light bulb jumps around. The light bulb is represented by an *ImageSprite* that is contained in the Canvas component. We’ll come back to this. The *Clock* is used as a timer to move the ImageSprite to a random spot on the Canvas every time the Clock ticks. The *Sound* is used to vibrate the phone or play a sound whenever the player hits the ImageSprite.

## Adding the Canvas

To begin with, set the various *Screen1* properties as shown in the UI screenshot above, including *AppName, ScreenOrientation, Title* and others. Next, drag and drop a *VerticalArrangement* component from the *Layout* drawer and set its height and width to *Fill parent.* The Canvas and other component will be contained within this arrangement and will fill the screen. By using a VerticalArrangement, we can guarantee that all of its components will be visible on the screen without scrolling.

You have used the Canvas component for drawing in the Paint Pot tutorials. In this app Canvas is used to support animation. App Inventor uses two types of components that can move for animation, *Balls* and *ImageSprites*, both of which are contained within a Canvas component. First, we’ll add the Canvas component.

1. Drag and drop a Canvas component from the Palette’s *Drawing and Animation* category into the *VerticalArrangement.* .
2. Set the Canvas’s *width* and *height* to *fill-parent*.
3. Set the Canvas’s *BackgroundImage* to *kitchen-picture.jpg*.

## Adding the Save Electricity Label

Add the *Label* to the UI, right underneath the *Canvas* in the *VerticalArrangement.* Name the label *LabelSaveElectricity* and set its *Text* property to “Help Save Electricity!”

## Adding the ImageSprite

As mentioned before, the light bulb is represented by an [ImageSprite](http://ai2.appinventor.mit.edu/reference/components/animation.html#ImageSprite), a component that can move (like a Ball) and can also display an Image (unlike a Ball). ImageSprites can only be used on a Canvas.

1. Drag and drop an *ImageSprite* component from the Palette’s *Drawing and Animation* drawer onto the *Canvas*.
2. Set the ImageSprite’s Picture property to the *lightbulb.jpg* image file, which is included in the template’s Media panel.
3. Set the ImageSprit’s Height and Width properties to 50 pixels each.
4. Rename the ImageSprite to “LightBulbSprite”.

## Adding the Sound

We will use a *Sound* Component to vibrate the device when the player successfully touches the light bulb. Drag and drop a *Sound* component from the Palette’s *Media* category into the Viewer. It will be named Sound1 and will appear as a non-visible component.

## Adding the Clock

The ImageSprite component has a Speed property, which controls the Sprite’s movement. However, for this app, we won’t be using that property. Instead, we will use the [Clock](http://ai2.appinventor.mit.edu/reference/components/sensors.html#Clock) component to move the Sprite to a random location on the canvas whenever the clock ticks. The Clock has a *Timer* event that can be used to move the Sprite to a new random location whenever the Timer event fires. In effect, every time the Timer ticks, the Sprite will ‘jump’ to a new random location.

1. Drag and drop a Clock component from the Palette’s *Sensors* category into the Viewer. It will be named *Clock1* and it will appear at the bottom of the Viewer as a *non-visible component.*
2. Set the Clock’s *TimerInterval* property to 1000 milliseconds. This will make the clock tick every second.

## Test the App as you Build It

As you build this app, test it by connecting App Inventor to your device (phone or tablet):

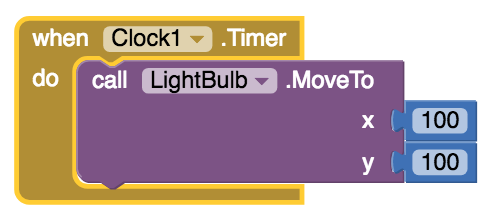
* In your browser, click **Connect** and select the ***AI Companion***.
* If the app doesn’t start downloading to the device almost immediately:
  + Make sure both your laptop and device are on the same Wifi network.
  + Try refreshing the App Inventor page in the browser and try again.

# Coding the App’s Behavior

## In order to get the Sprite moving, we’ll first set up a *Clock.Timer* event that will allow the Sprite to ‘jump’ to various positions randomly with the help of some Math blocks that are used for randomness. Once the Sprite is moving we’ll need to make the device vibrate each time the user touches the Sprite (light bulb).

## Moving the Sprite

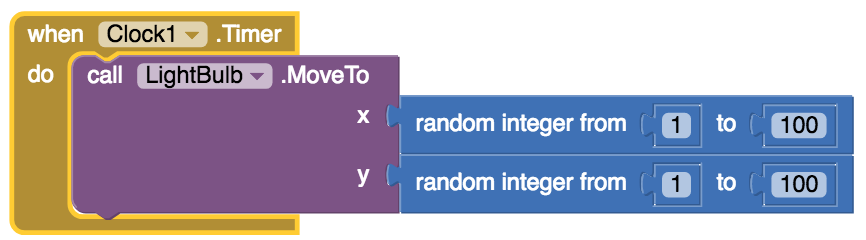
Consider the following blocks:



To construct this you will pull the when *Clock1.Timer* block from the Clock1 drawer, the *MoveTo* block from the *LightBulb* drawer, and two number blocks from the *Math* drawer.

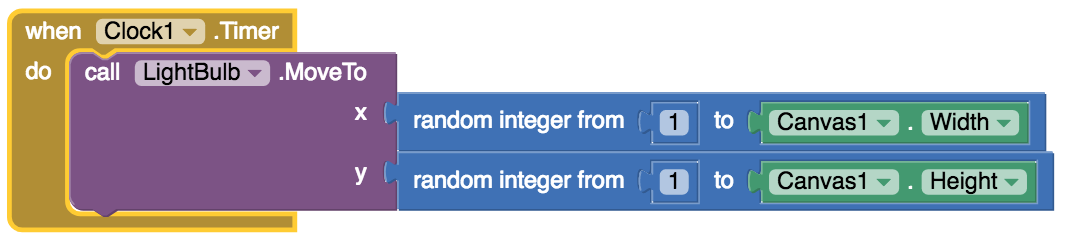
The *Clock.Timer* event is triggered periodically. If you have set the Clock’s *TimerInterval* property to 1000, this event will trigger every 1000 milliseconds, or every second. If you are testing the app, you’ll see that the Sprite jumps to the location x=100, y=100 and just stays there. The event keeps triggering, but after the first time the Sprite is already at 100,100 so it doesn’t appear as if anything is happening.

What we really want is for the Sprite to move to a random location each time the clock triggers. For this, drag out two random integer blocks from the Math drawer, and plug them in as follows:



Now you should see the Sprite move around randomly every 1000 ms., but stay within the top left section of the Canvas.

Of course the upper limit of 100 is incorrect. We really want the Sprite to jump around anywhere on the Canvas. For this, we can use the Width and Height properties of the Canvas. Pull out *Canvas.Width*, *Canvas.Height* from the *Canvas* drawer and modify your blocks as follows:



Now you should see the LightBulb jumping randomly all over the canvas.

## Giving the Sprite a Random Starting Place

When the app starts, and even before the clock ticks start, we should place the Sprite randomly. We can do this by pulling out a *Screen.Initialize* block from the *Screen* drawer, copying the blocks within the *Clock.Timer* event, then pasting them in *Screen.Initialize*:

## ScreenInitialize.png

## Refactoring: The *moveRandom* Procedure

As we learned in an earlier lesson, having identical segments of code in two separate locations in the app is not a good programming practice. To fix that, let’s define a *moveRandom* procedure and call it to manage moving the ImageSprite to a random location.

## MoveRandomProcedure.png

**Abstraction. When you define a procedure in a programming language you are creating an abstraction.** The procedure ***represents*** a particular algorithm. Once you define the procedure, it encapsulates the details of the algorithm. To execute the algorithm, you only need to ***call the procedure.*** When calling the procedure, you aren’t necessarily aware of the details of the algorithm. Thus, defining and using procedures helps **reduce the complexity in our programs** and makes them easier to read and modify and maintain.

In our app, the *Screen.Initialize* and *Clock.Timer* events both call the same blocks to move the ImageSprite randomly. Whenever you copy-paste, as we did, a little bell should go off in your head -- I need a procedure! In this case, the moveRandom procedure is appropriate.

1. Drag out a to procedure block from the Procedures drawer.
2. Name the procedure “moveRandom”.
3. Drag the blocks from *Screen.Initialize* and place them in “moveRandom”.
4. Open the Procedures drawer again. You should now see a block, “call moveRandom”. Pull in out and place it in *Screen.Initialize*. Take out the blocks for *Clock.Timer* as well and replace them with a call moveRandom block.

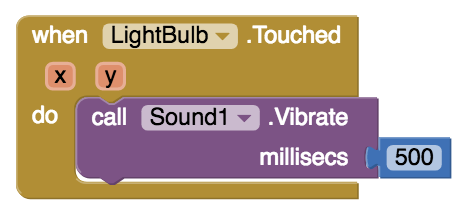
Your app should work exactly the same, with the Sprite moving around randomly.

## Handling the Touched Event

Similar to Paint Pot, the *Turn Off Lights* app will also use a Touched event. However, this touched event is used differently. When a player touches the Sprite we want them to get some type of feedback for their accomplishment. This helps give the user feedback when taking actions within the app. For now, let’s make the phone vibrate for 500 milliseconds (half a second) whenever the Sprite is touched:

1. Get a *LightBulb.Touched* block from the *LightBulb* drawer.
2. Get a *Sound1.Vibrate* block from the Sound1 drawer.
3. Set the *Vibrate* interval to 500 millisecs.
4. Place the *Vibrate* block inside the *Touched* block.

The finished block should look like this.



Test the app’s behavior by trying to touch the *moveRandom* as it jumps around. Does the device vibrate when you touch it?

**NOTE** that not all Android devices contain a vibration component. If your device does not vibrate, replace the *Vibrate* block with a *Sound1.Play* block and set the *Sound1.Source* property (in the Designer) to the *beep.mp3* file.

# Reflection for the Student

In your portfolio, create a new page named ***TurfOffLights*** and answer the following questions:

1. This app presents a new type of event which you haven't encountered before. What is that new event? How often is it triggered?
2. Consider the apps you've developed so far. Can you list all the different events your apps have responded to? What other events do you think an app can respond to? Explore some of the components in App Inventor and see what event handlers they have.
3. What are the advantages of writing procedures in programming? Consider the procedures you wrote for this app.

# Exercises

To practice your skills and test your knowledge try some [interactive exercises](https://course.mobilecsp.org/mobilecsp/unit?unit=23&lesson=53).