# Table of Contents

**Animation**
- Canvas Coordinate System
- Creating a Canvas
- Painting on a Canvas
- Creating and Moving Image Sprites and Balls
  - Timer Events
    - High-Level Animation Functions
      - Edge Reached
      - CollidingWith and NoLongerCollidingWith
    - Interactive Animation
      - Stopping and Starting By Disabling and Enabling Timers
**Summary**
Animation

This chapter discusses methods for creating apps with simple animation—objects that move. It begins by describing the Canvas component and its x-y coordinate system, then explains how timer events form the basis of animated behavior, and finally discusses the high-level functions and event-handlers, such as CollidedWith, provided in the Image Sprite and Ball Component.
Animation is the rapid display of images to create the illusion of movement. Animation can be created with something as primitive as a flipbook to some of the most sophisticated hardware and software systems invented.

**Canvas Coordinate System**

In App Inventor, animation is specified within a Canvas component. A Canvas component is used for either end-user drawing, such as in the PaintPot program, or for animation, such as in the Mole Mash app. All drawing and animation within an App Inventor app must occur within a Canvas.

The Canvas consists of pixels, the tiniest dots of color that can appear on a phone (or other device). The Canvas is defined by an X-Y coordinate system, with x defining a location on the horizontal plane (column), and y defining a location on the horizontal plane. As you move to the right in the Canvas, the x-coordinate is larger. As you move down, the y-coordinate gets larger:

<table>
<thead>
<tr>
<th>x=0</th>
<th>x=1</th>
<th>x=2</th>
<th>x=3</th>
<th>x=4</th>
<th>x=5</th>
<th>x=6</th>
<th>x=7</th>
<th>x=8</th>
<th>x=9</th>
<th>x=10</th>
<th>x=11</th>
<th>x=12</th>
<th>x=13</th>
<th>x=14</th>
<th>x=15</th>
<th>x=16</th>
<th>x=17</th>
<th>x=18</th>
<th>x=19</th>
</tr>
</thead>
<tbody>
<tr>
<td>y=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y=2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y=3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y=4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y=5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y=6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y=7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y=8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y=9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y=10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y=11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y=12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y=13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y=14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Computer scientists like to count beginning with 0, so the top-left cell in a Canvas is (x=0,y=0). The cell to its right is (x=1,y=0). The top-right corner has an x-location the same as the width of the canvas minus 1. Most phone screens have a width close to 300, but for the sample canvas shown here, the Canvas width is 20 so the top-right corner is the coordinate (x=19,y=0). Note that the Canvas component has a Width and Height property.
Creating a Canvas

The Canvas component is in the Basic palette. After you drag one into an app, you can specify its width and height. Often, you'll want the width to span the width of the device screen. To do this, choose Fill Parent when specifying the width:

![Canvas Width Setting](image)

You can do the same for the height, but often you'll set it to some number (e.g., 300 pixels) and leave room for components above and below the Canvas.

Painting on a Canvas

An app can paint on the canvas using two primitive drawing functions, DrawLine and DrawCircle. For example, the blocks:

![Drawing on Canvas Block](image)

draw a line of length 5 from the top-left toward the bottom-right corner (diagonally)

Creating and Moving Image Sprites and Balls

You can also add ImageSprite and Ball components to a canvas. Both are objects that can be moved around in a canvas. The Ball is a circle—you can only change its appearance by changing its color or radius. The ImageSprite behaves the same as a Ball, but it can take on the appearance of any image you load. Image Sprites and Balls can only be added within a Canvas—if you try to drag them into the user interface outside of a Canvas, they will not be added.

Both image sprites and balls can be moved within the canvas with the MoveTo operation, e.g.,
move the image sprite so that its top-left corner is at coordinate (x=5, y=5):

**Timer Events**

Apps consist of events and event-handlers, and probably the most common type of event is user-initiated, e.g., the user clicking a button.

With App Inventor, you can also define *timer events*, and these events for the basis of computer animation. With App Inventor, all animation is essentially image sprites (and balls) moving in response to timer events.

To define a timer event, you first drag a Clock component into your app within the Component Designer. The Clock component has a property TimerInterval associated with it. The interval is defined in terms of milliseconds (1/1000 of a second). If you set the TimerInterval to 500, that means a timer event will be triggered every ½ second.

In the Blocks Editor, you can define animation by specifying that an object’s location changes every timer event. App Inventor only provides a MoveTo function, not a MoveBySomeAmount function. With MoveTo, you can specify that an object moves some amount from its current location by adding (or subtracting) from its current location.

For instance, if you want an object to move horizontally across the screen, you would call MoveTo within a timer event and set the x-slot to the object’s previous location plus some number, leaving the y-slot at its current location:
If you wanted to move an object diagonally, you’d add to both the x- and y-coordinate:

**High-Level Animation Functions**

App Inventor provides some high-level blocks for handling game-like situations like when an object reaches the edge of the canvas or collides with another object. If you had to, you could such situations by programming a timer-event to check the coordinates of the image sprites. Fortunately you don’t need to as the high-level functions can make things easy.

**Edge Reached**

Consider again the animation in which the object is moving diagonally from the top-left to the bottom-right of the canvas. Often with such animation, you’ll want the object to reappear somewhere else on the canvas instead of disappearing when it reaches the edge of the canvas.

You can program such a behavior easily using the EdgeReached event within a Ball or ImageSprite object:
EdgeReached is triggered when the Ball hits any edge (top, bottom, left, or right). This event handler, combined with the diagonal movement specified with the timer event above, will cause the ball to, continuously, move left-top to bottom-right, hit the bottom (or right) edge, then pop back up to the top-left.

**CollidingWith and NoLongerCollidingWith**
Shooting games, sports, and other animated apps often rely on activity occurring when two or more objects collide, e.g., the bullet hitting the target.

Suppose, for instance, that besides the ball moving diagonally there was another ball moving around in the canvas. Suppose also that you wanted something to happen when the two objects collided, say have the first ball pop back up and left some amount. Here are the blocks for such an event-handler:

With NoLongerCollidingWith, you could program a behavior in which an object changes its appearance while touching another object, then changes back once it is no longer touching.

**Interactive Animation**
With the animated behaviors discussed so far the end-user isn’t involved. Of course for interactive games, the end-user plays a central role. Often the end-user will control the speed or direction of an object with buttons or other user interface objects.

**Stopping and Starting By Disabling and Enabling Timers**
As a first example, let’s update the diagonal animation by allowing the user to stop and start the diagonal movement. One way to do this would be to program a Button.Click event-handler to disable and re-enable the timer event of the clock component.

By default, the Clock component’s timerEnabled property is checked. You can disable it dynamically by setting it to false in an event-handler. For instance, suppose you wanted to program a button that on first click stops the movement of a ball:

![Event handler diagram]

After the Clock1.TimerEnabled property is set to false, the Clock1.Timer event will no longer trigger, and the ball will stop moving.

You could “toggle” the movement of the ball by adding an if-else in the event-handler that either enables or disables the Timer:

![Event handler diagram]

This event-handler stops the timer on first click, and resets the button so that it says “Start” instead of “Stop”. The second time the button is clicked, the TimerEnabled is false, so the else
part is executed. In this case, the timer is enabled, which gets the object moving again, and the button text is switched back to “Stop”.

**Summary**

App Inventor facilitates simple animation with the Canvas, ImageSprite, Ball and Clock components, and some high-level functions for handling collisions and an object reaching the edge of a canvas.

Animation is defined using timer events that trigger object movement and the MoveTo function. You can control the speed of an object by modifying the TimerInterval of the Clock component or moving further each timer event. You can change the direction of an object by modifying its x- and y-locations by more or less on each timer event.