CHAPTER 9

Graphics and Animation

Topics

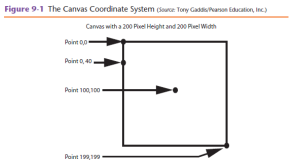
- The `Canvas` Component
- The `Ball` and `ImageSprite` Components
- Using the Clock Component to Create Animations
- Dragging sprites
- Detecting Collisions

The `Canvas` Component

- The `Canvas` is a component that allows us to create two-dimensional graphics.
- We can also use `ImageSprites` on the `Canvas`.
- The `Canvas` Component is the starting point for creating graphics and animations.
- The `Canvas`, found in the *Drawing and Animation Pallet*, is like a sub-panel inside the `Screen` Component.

The `Canvas` Component

- The X coordinate is represented by the number of pixels to the right of the left edge.
- The Y coordinate is represented by the number of pixels down from the top edge of the canvas.



The `Canvas` Component

Canvas Properties

- The `Height` and `Weight` properties are important because they set the size of the canvas.
- `PaintColor` sets the color for the points, lines and circles.
- `BackgroundColor` will fill the entire background of the canvas.
- `BackgroundImage` sets the image in the background. First we must upload an image to our app.

The `Canvas` Component

Drawing Methods

- Once you add a canvas component, you will find method blocks in the Blocks Editor that will allow you to draw on the canvas. It is found in the *Canvas1* drawer.
- `DrawPoint` – will draw a one pixel point on the canvas.

The `Canvas` Component

Drawing Methods

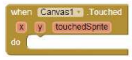
- `DrawCircle` – will draw a circle on the canvas. The circle is drawn filled in with the color specified by the `PaintColor` properties value.
- `DrawCircle` – requires you give it a radius in pixels.
- `DrawLine` – will draw a line in the color specified by the `PaintColor` property.
- `Clear` – clears all graphics from the canvas except the background image.

The `Canvas` Component

Touch and Drag Events

- The Canvas components `Touched` event will activate when a user touches the canvas.
- If the canvas has a sprite and the sprite is touched, this event will set the `touchedSprite` pramater to true.

Figure 9-2 Touched Event Handler (Source: MIT App Inventor 2)



The `Canvas` Component

Touch and Drag Events

- The `Dragged` event will trigger when a user drags across canvas.
- This event will record where the drag started and where it ended by keeping track of the X and Y coordinates.
- There is also a Boolean variable `draggedSprite` that is set to true if the user drags a sprite on the canvas.

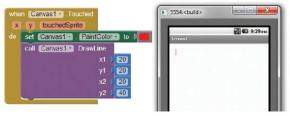
Figure 9-3 Dragged Event Handler (Source: MIT App Inventor 2)

The `Canvas` Component

Drawing Using Specific Values

- You can hard code or use variable data to draw on a canvas.
- To draw a line from the coordinates 20, 20, to 20, 40 when the user presses a button, supply the `DrawLine` method with those specific values as shown in Figure 9-12.

Figure 9-12 Drawing with Hard Coded Values (Source: MIT App Inventor 2)

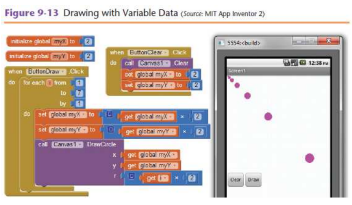


The `Canvas` Component

Drawing Using Specific Values

You can also use variable data with the drawing methods.

Figure 9-13 Drawing with Variable Data (Source: MIT App Inventor 2)



The `Canvas` Component

Drawing Using Specific Values

- In this `DrawCircle` example, we set up two variables for the X and Y coordinates.
- We use a variable `i`, the `for range` loop counter, for the radius.
- The result in the emulator (Figure 9-13) shows that the coordinates are variable.

The Ball and ImageSprite Component

- A sprite is a two-dimensional graphic, picture, or animation that can be moved about the canvas.
- App Inventor has two different types of sprite components.
  1. The `Ball` component, is essentially a two-dimensional circle.
  2. The `ImageSprite` component, acts very similar to the `Ball` component, except that you select an image for it to display.

The Ball and ImageSprite Component

The `Ball` and `ImageSprite` Component Properties:

- The sprite components have X and Y properties that indicate the sprites position on the canvas.
- The X and Y coordinates represent the top left corner of the Sprite.

The Ball and ImageSprite Component

The `Ball` and `ImageSprite` Component Properties

•Sprite components also have properties that tell it how to behave. They are summarized here:

•`Interval` – sets how often this sprite will move in milliseconds.

•`Speed` – the number of pixels to move each interval.

The Ball and ImageSprite Component

The `Ball` Component Properties

You can change how the `Ball` sprite appears.

- `PaintColor` – to set the color.
- `Radius` – to set the size.
- `Visible` – to either show or hide it.

The Ball and ImageSprite Component

The `Ball` and `ImageSprite` Component Methods

- Sprite components have a `Bounce` method that simulates the sprite bouncing off an edge or corner.
- It is important to understand *Edges* before using this method.

The Ball and ImageSprite Component

Edges

Edges are represented by numbers and are used in the `Bounce` method and the `EdgeReachedEvent`. The edges are represented as follows:

North or Top Edge = 1	Northeast Edges = 2
South or Bottom Edge = -1	Southwest Edge = -2
East or Right Edge = 3	Southeast Edge = 4
West or Left Edge = -3	Northwest = -4

The Ball and ImageSprite Component

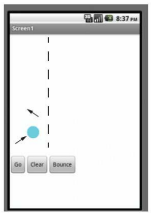
Edges

- Assume we are using a `Ball` component and the *Bounce* button shown in Figure 9-14 called the `Bounce` method with a number 3 as the edge parameter.
- At the point in which it is invoked, the bounce method will act as if it were an East edge to bounce off of (represented by the dotted line).

The Ball and ImageSprite Component

- The `MoveTo` method will allow you to move the sprite so a specific X, Y coordinate on the canvas.
- Sometimes sprites may accidentally move out of bounds, use the `MoveIntoBounds` method to put them back onto the canvas.

Figure 9-14 Bounce with East Edge (Source: MIT App Inventor 2)



The Ball and ImageSprite Component

The `Ball` and `ImageSprite` Component Events

- The *`Dragged`* event has the same arguments and works the same as the canvas *`Dragged`* Event.
- This event will be triggered when a user drags the sprite with their finger or mouse.
- It holds these values in the `prevX`, `prevY`, `currentX`, and `currentY` argument values.

The Ball and ImageSprite Component

The `Ball` and `ImageSprite` Component Events

- The `EdgeReached` event is called when the sprite reaches an edge of the canvas.
- Like the canvas `Touched` event, the `SpriteTouched` event will execute when the Sprite is touched. It will record the X, Y coordinate positions where the touch occurred.

Figure 9-15 `EdgeReached` Event Handler (Source: MIT App Inventor 2)

The Ball and ImageSprite Component

- The `Ball` and `ImageSprite` Component Events
- A fling is a quick swipe of the canvas and will invoke the `Flung` event.
- The `Speed` and `Heading` values of the flung `Ball` are also available.

Figure 9-16 `Flung` Event Handler (Source: MIT App Inventor 2)

Using the `Clock` Component to Create Animations

- The `Clock` Component is a timer that fires an event based on a time interval.
- The `Clock` Component has three properties and one event, the *`Timer`* event.
- The `Clock` has many methods available to show and manipulate dates.

Using the `Clock` Component to Create Animations

`Clock` Compliment Properties

- *`TimerEnabled`*– can be true or false. If set to true, the timer will fire on the set interval.
- *`TimerInterval`*– the interval for the timer. Its value is in milliseconds.
- *`TimerAlwaysFires`*– If set to true, it will fire even if the application is not active on the screen and is running behind the scenes on your device.

Using the `Clock` Component to Create Animations

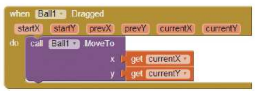
`Clock` Timer Event

The `Timer` the event fires at each interval specified by the `TimerInterval` property.

Dragging Sprites

To drag a sprite, use the `Dragged` event in conjunction with the `MoveTo` method.

Figure 9-33 Using the `Dragged` Event Handler and the `MoveTo` Method Block
(Source: MIT App Inventor 2)



Dragging Sprites

- Figure 9-33 shows a Ball sprite's `Dragged` event handler used with the `Ball1.MoveTo` method.
- The `Dragged` event of a sprite will keep track of where the drag started with the `prevX` and the `prevY` values.
- Use the `currentX` and `currentY` values to move the ball as the drag occurs.

Detecting Collisions

- You can detect when a sprite collides with another component.
- The App Inventor sprite components have `CollidedWith` and `NoLongerCollidingwith` event handlers.
- The `CollideWith` event handler will fire when any collision happens.
- You must have an `if then` condition inside the event to determine what it collided with.
- In the My Blocks panel, each component's ***last*** block is the `component` block.

Detecting Collisions

Assume we have an `ImageSpriteBalloon1` component (Figure 9-42).

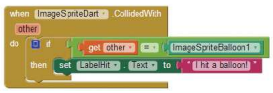
Figure 9-42 Component Block for `ImageSpriteBalloon1` (Source: MIT App Inventor 2)

This is the block we will use to determine which component was hit in the collision.

Detecting Collisions

- In Figure 9-43, you see that the `ImageSpriteDart.CollidedWith` event has an argument *other*.
- Compare this block's value with the component `ImageSpriteBalloon1` block.

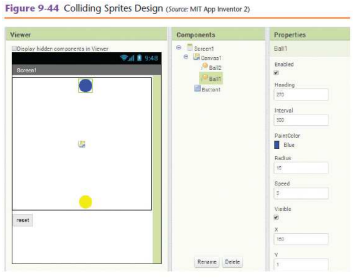
Figure 9-43 `CollidedWith` Event (Source: MIT App Inventor 2)



Detecting Collisions

Let's make two sprites collide.

Figure 9-44 Colliding Sprites Design (Source: MIT App Inventor 2)



Detecting Collisions

- We have two Ball sprites, one blue and one yellow.
- They each have and `Interval` of 500, `Radius` of 15, and a `Speed` of 5.
- They are lined up together on the x coordinate, a value of 150.
- The `Canvas` size is set to fill the parent for the `Width`, and its `Height` is 300 pixels.

Detecting Collisions

- The blue ball will travel straight down. `Heading` property is set to 270. The yellow ball will travel straight up, its `Heading` is set to 90.
- We will program the application to turn both balls green when the collision occurs.

Detecting Collisions

Use the `Ball1.CollidedWith` event handler to change the balls to green and switch their direction as in figure9-45.

Figure 9-45 Programming the Collision (Source: MIT App Inventor 2)