COLLEGIO TECHNOLOGIES INC.

REACT NATIVE: ANIMATIONS AND GESTURES

INTRODUCTION

- We now have all of the basic building blocks in play for creating a React Native app, so now it's time to dig into some of the cooler features!

- We'll start by taking a look at how we can really customize our apps by digging into animations and gestures!

REACT NATIVE ANIMATIONS

- So far, we've seen an example of animation in our apps when we used **LayoutAnimation**.
    - This is a fantastic method for creating simple, easy animations, but it doesn't allow you much control over the animation itself.
    - In addition, due to the lack of control, you may be stuck with animation on some elements that don't necessarily require it.
- In addition to **LayoutAnimation**, we also have the **Animated** module available for our use.
    - Although it is more complicated to set up than LayoutAnimation, it also allows for more complex animations.
    - More control means a better experience for the user, especially when gestures are involved.

THE ANIMATED MODULE

- The **Animated** module allows you to create custom animations within your app.
  - As such, it comes with a variety of components, objects and functions with properties. This is where the complexity comes into place.

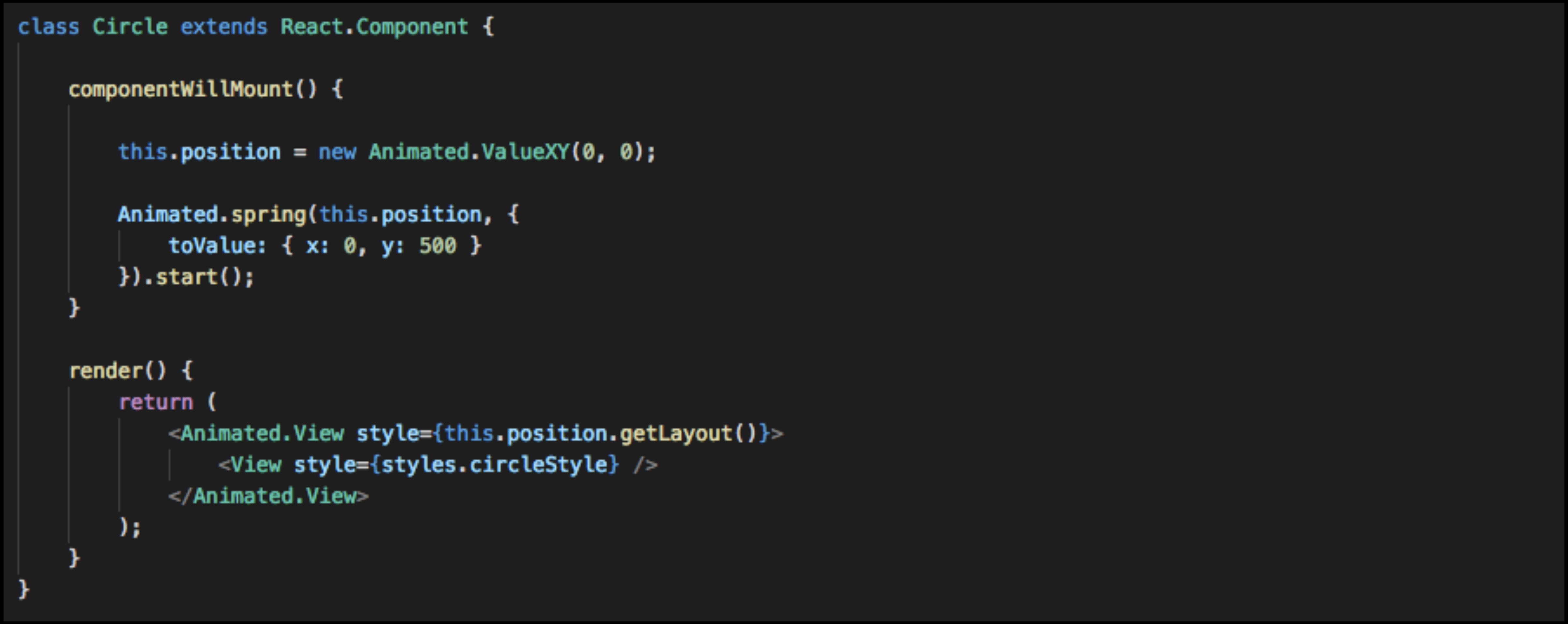
- In order to create any animation, you should know:
  - Which element is being animated?
  - What is the status of the animation at time **x**?
  - Where is the element being moved to?

COMMON ANIMATED MODULE TYPES AND VALUES

- **Types (Animated.Types)** - controls how the animation is changing, and where the component is moving to.
  - **Spring**
  - **Decay**
  - **Timing**
- **Values (Animated.Values)** - Used to declare the position of the element.
  - **Value**
  - **ValueXY** - used to show and set the current location of the animated component.
- **Components (Animated.Components)** - Where the animated elements will live.
  - **View**
  - **Text**
  - **Image**

THE ANIMATED MODULE

- You can import the animated module by typing:
  - **import { Animated } from 'react-native';**
- By default, all animations are 1 second.
- We animate our objects by passing the **getLayout()** method of the **Animated.ValueXY** object to our animated component's style.
  - **Please note**, components in the Animated module are completely different from standard components.



```
class Circle extends React.Component {

    componentWillMount() {

        this.position = new Animated.ValueXY(0, 0);

        Animated.spring(this.position, {
            toValue: { x: 0, y: 500 }
        }).start();
    }

    render() {
        return (
            <Animated.View style={this.position.getLayout()}>
                <View style={styles.circleStyle} />
            </Animated.View>
        );
    }
}
```

THE ANIMATED MODULE WORKFLOW

- Animations performed using the Animated module are performed completely outside of state.
- Instead, the workflow for using animations is as such:
  - Render the **Animated.View** and elements inside of it
  - **Animated.View** finds the animated value via it's own props (**getLayout()**).
  - The **ValueXY** object values start changing.
  - The **Animated.View** sees the updated value from the **ValueXY** object.
  - The View updates it's styling based on the changed **ValueXY** object.

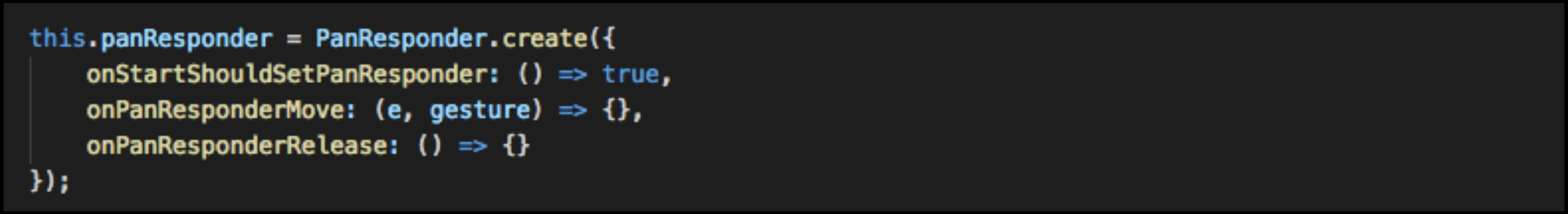
GESTURES WITH PAN RESPONDER

INTRODUCTION TO GESTURES

- In terms of mobile usage, a gesture is considered to be any movement that you perform with your hand on the screen, such as a swipe, pinch or tap.
- We will use the **PanResponder** module to handle gestures that are made on the screen.
- Like with the **Animated** module, the **PanResponder** system requires three requirements to be met:
  - What component are we interacting with?
  - What component is handling the gesture?
  - How are we changing the display with the gesture?
- You will need to create and configure a separate **PanResponder** instance for each individual gesture that you want to handle on the screen.
- **PanResponder** instances are typically created inside of the component **constructor()**.

PAN RESPONDER PROPERTIES

- **onStartShouldSetPanResponder()** - this is invoked every time the user taps or presses on the screen. By setting this property as a function and returning **true**, you are telling the instance of the Pan Responder to trigger whenever the gesture is used.
  - This is an arrow function because we can add some conditional logic to determine if we should use the Pan Responder or not.
- **onPanResponderMove()** - this is invoked whenever a user is moving their finger around the screen.
  - The callback has two arguments: the event that is triggered, and the gesture data itself.
- **onPanResponderRelease()** - this is called whenever a user releases their finger from the screen.
  - Also contains an event and gesture data argument

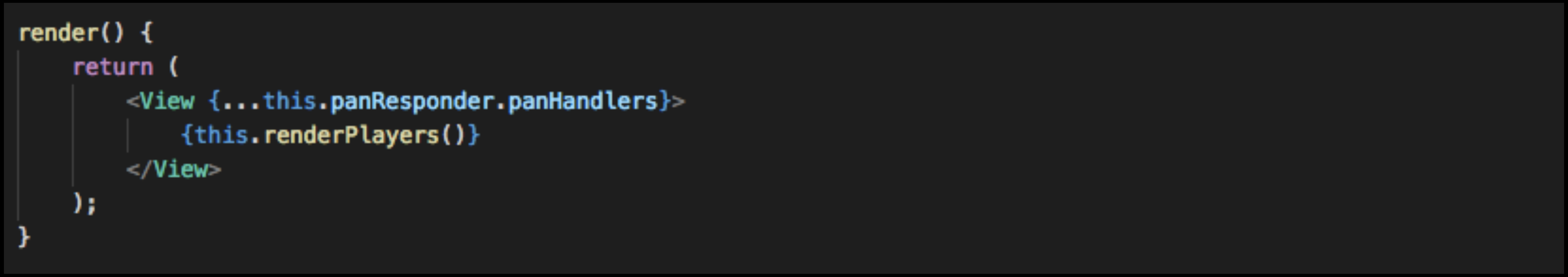


```
this.panResponder = PanResponder.create({
    onStartShouldSetPanResponder: () => true,
    onPanResponderMove: (e, gesture) => {},
    onPanResponderRelease: () => {}
});
```

PANRESPONDER.PANHANDLERS

- You use the **PanResponder.panHandlers** to intercept presses and taps from the user.
  - You can tie this into the element that you want to apply the gesture to by adding the **PanResponder.panHandlers** attribute using the spread operator.

- Once we have our panHandlers added, we can set the gesture to respond with an animation by adding the **getLayout()** function to the style of the **<Animated.View>** tag.



```
render() {
    return (
        <View {...this.panResponder.panHandlers}>
            {this.renderPlayers()}
        </View>
    );
}
```

THE GESTURE OBJECT

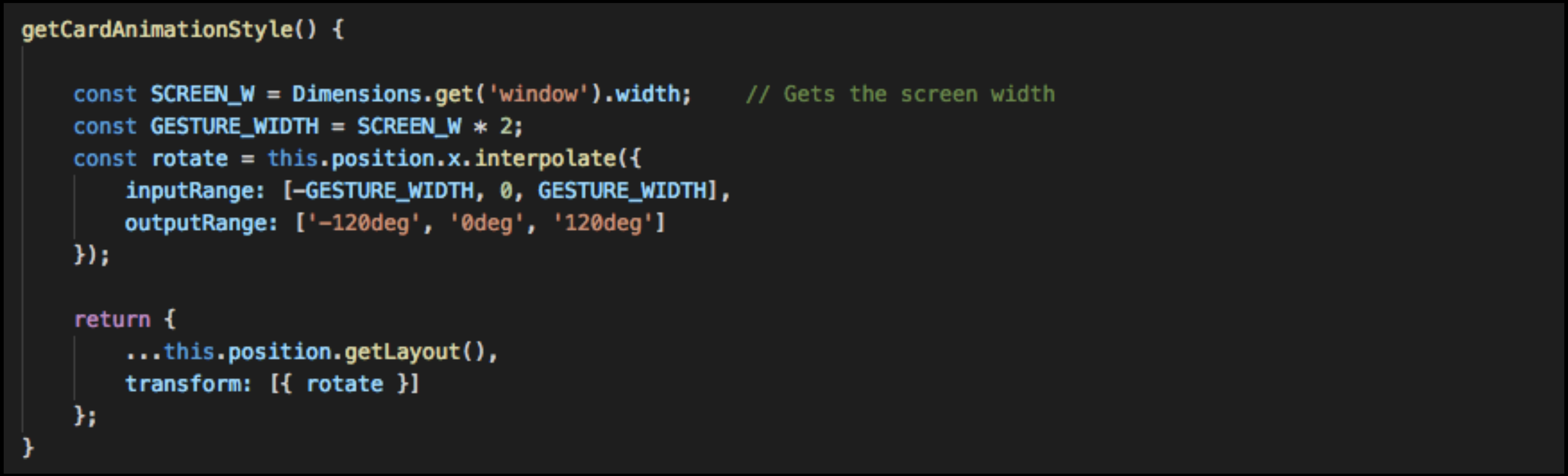
- The gesture object that you have access to in your **onPanResponderMove()** function will contain many different values pertaining to the actual movement itself, such as:
  - **dx:** the distance along the x-axis that your finger travelled in total for the gesture.
  - **dy**: the distance along the y-axis that your finger travelled in total.
  - **moveX:** where along the x-axis the user's finger is
  - **moveY:** where along the y-axis that the user's finger is
  - **vx:** how quickly along the x-axis that the user is moving
  - **vy:** how quickly along the y-axis that the user is moving
  - **numberActiveTouches:** how many fingers are touching the screen (used for double finger functionality)

INTERPOLATION

- We can use interpolation to perform a linear change on our animated objects, based on the x or y axis.
  - Animation can include things such as fading, color change, sizing, and rotation.

- The interpolate function accepts an object that has two properties:
  - **inputRange** - the range of values that declare the bounds of the gesture
  - **outputRange** - the interpolated value that will be applied to the element
  - The above values are used to create a linear relationship to the position of the selected gesture.
  - We can use the **Dimensions** module to get the screen width/height for gestures.

- Along with the position of the element, we return the change in the style property of the selected **Animated.Container**.



```
getCardAnimationStyle() {

    const SCREEN_W = Dimensions.get('window').width;    // Gets the screen width
    const GESTURE_WIDTH = SCREEN_W * 2;
    const rotate = this.position.x.interpolate({
        inputRange: [-GESTURE_WIDTH, 0, GESTURE_WIDTH],
        outputRange: ['-120deg', '0deg', '120deg']
    });

    return {
        ...this.position.getLayout(),
        transform: [{ rotate }]
    };
}
```

EXERCISE: FREE AGENT TRACKER (TINDER EDITION!)

- Time to use animations and gestures to give the free agent tracker a ‘tinder-like’ feel to it. You’ll be doing the following:
  - Pull in a list of free agent data containing names, messages, ids and thumbnail_images
  - Display the currently selected player inside of a card. Allow the option to swipe the card to the left or right. The swipe motion should have a threshold, so that when it gets past a certain point, it keeps going.
  - Use interpolation to rotate your cards as you swipe them off the screen.
  - Print a message to show that you are properly handling the swipe to the left or right.

RESOURCES

- **React Native Animated Module:**
  - **Official:** https://facebook.github.io/react-native/docs/animated.html
- **React Native PanResponder Module:**
  - **Official Site:** https://facebook.github.io/react-native/docs/panresponder.html

DISCLAIMER