

# Introduction to JavaFX

Effects, Animation & Production Suite

March 18, 2009

Jim Connell

[jconnell@schematic.com](mailto:jconnell@schematic.com)

# Vibe

Partnered with Sun @

- JavaFX Launch
- Mobile World Congress
- SXSW



# Topics

- 5 minute JavaFX script introduction
- Transforms & Effects
- Animation
- Events
- Production Suite

# JavaFX Script Introduction

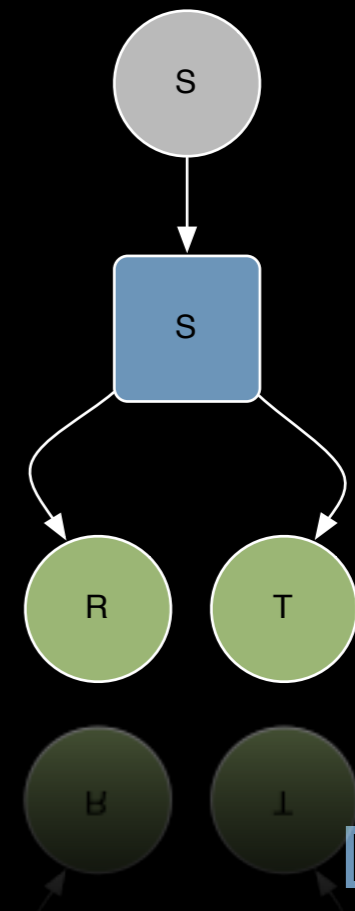
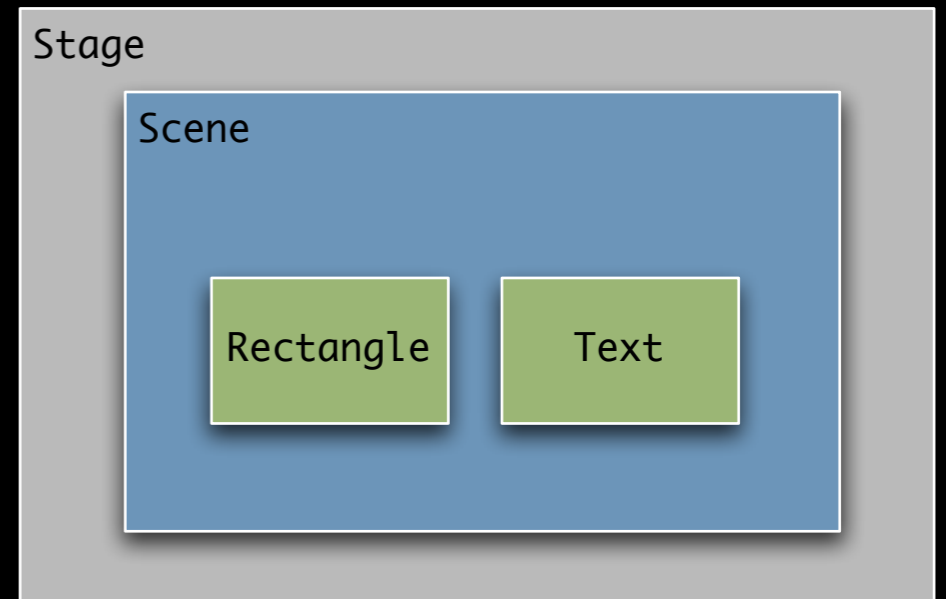
# Your First JavaFX Script

```
Stage {  
  title: "Hello JavaSIG", width: 400, height: 600  
  scene: Scene {  
    content: [  
      Rectangle {  
        x: 15, y: 10  
        width: 250, height: 100  
        stroke: Color.BLACK  
        fill: Color.LIGHTGREEN  
      },  
      Text {  
        font : Font {  
          size : 36  
        }  
        x: 20, y: 45  
        content: "Hello JavaSIG"  
        fill: Color.WHITE  
        strokeWidth: 2  
        stroke: Color.DARKGREEN  
      }  
    ]  
  }  
}
```



# First Program 2nd Look

```
Stage {
  title: "Hello JavaSIG", width: 400, height: 600
  scene: Scene {
    content: [
      Rectangle {
        x: 15, y: 450
        width: 250, height: 100
        stroke: Color.BLACK
        fill: Color.LIGHTGREEN
      },
      Text {
        font : Font {
          size : 36
        }
        x: 20, y: 505
        content: "Hello JavaSIG"
        fill: Color.WHITE
        strokeWidth: 2
        stroke: Color.DARKGREEN
      }
    ]
  }
}
```



# What is a SceneGraph?

- Declarative method to defining visual structure
- A composite hierarchy of objects:
  - Node, a leaf (terminal node)
  - Group, a composite node (contains others)

# Items in 1st Program

Item	Description
Stage	Device independent container
Scene	The container for the <u>SceneGraph</u>
Rectangle	A node that knows how draw rectangles
Text	A node that draws Text

# Features of Nodes

- **Have one parent**
- **Transforms** (translation, scaling, etc)
- **Effects** (shadows, blurs, lighting, etc)
- **Event handlers** (keys & mouse)
- **Visual Properties** (coordinates, height, width, etc)

# JavaFX Script Features

- Supports OO constructs: classes, functions, members
- UIs are constructed using a Scene Graph
- Can call existing Java code
- Java can call FX with a little effort

# JavaFX Features (cont)

- Supports binding, change notifications propagated automatically
- No more property change listeners
- Supports triggers, code that gets executed when values change

# Transforms & Effects

# Structure of Examples

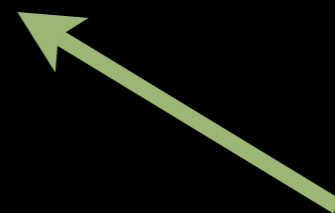
```
package presentation.transform;

import javafx.scene.transform.*;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.Rectangle;
import javafx.scene.paint.Color;
```

```
var rect = Rectangle {
    x: 100
    y: 10
    width: 200
    height: 200
    fill: Color.DARKOLIVEGREEN
};

var rotateRect = Rectangle {
    x: 100
    y: 250 // start lower than original
    width: 200
    height: 200
    fill: Color.DARKOLIVEGREEN
    transforms: [
        Rotate {
            angle: 45
            pivotX: 150
            pivotY: 350
        }
    ]
};
```

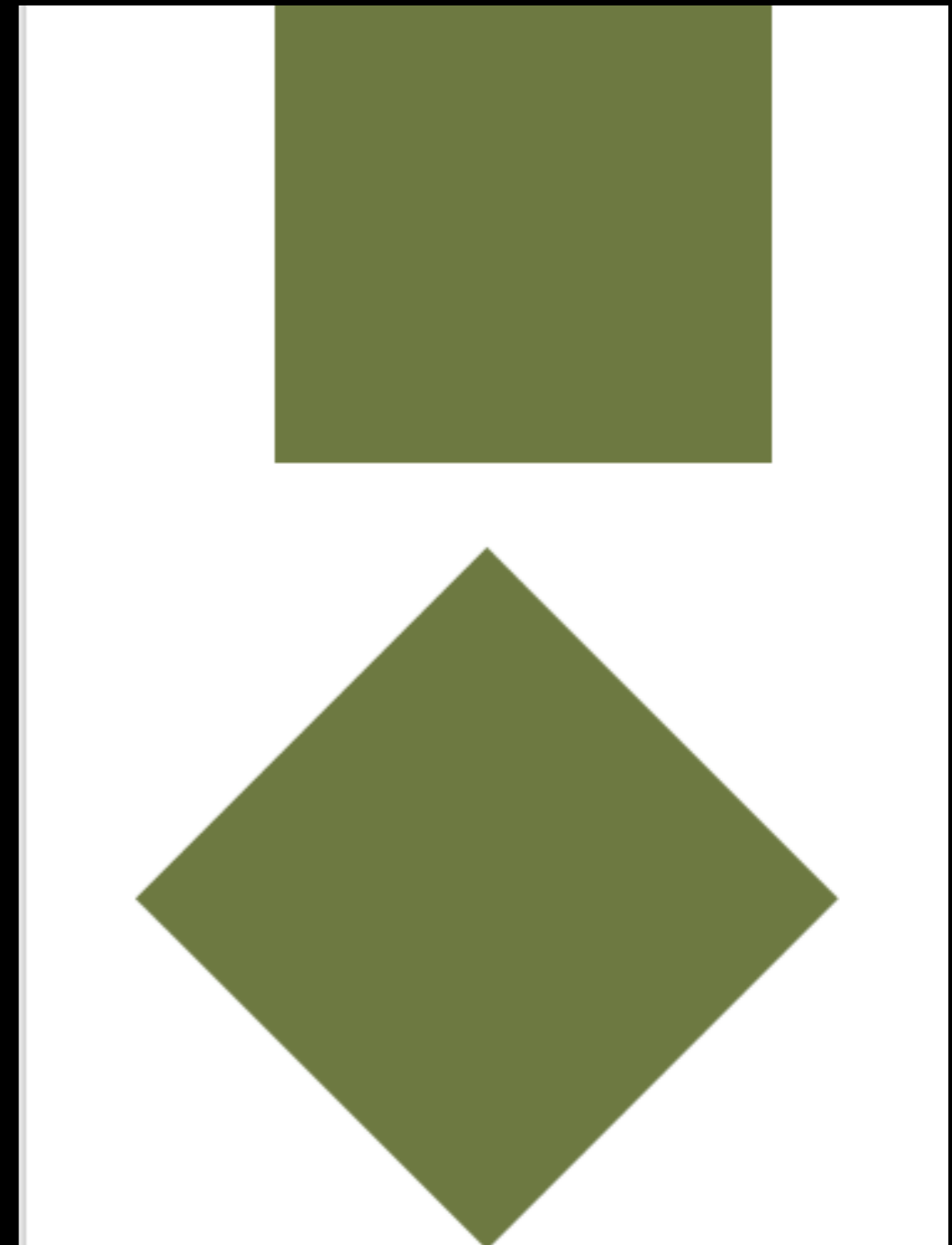
```
// continued from snippet on left
Stage {
    title: "Rotate"
    scene: Scene {
        width: 400
        height: 600
        content: [
            rect,
            rotateRect
        ]
    }
}
```



Examples will just show this

# Transform: Rotate

```
var rotateRect = Rectangle {  
  x: 100  
  y: 250  
  width: 200  
  height: 200  
  fill: Color.DARKOLIVEGREEN  
  transforms: [  
    Rotate {  
      angle: 45  
      pivotX: 150  
      pivotY: 350  
    }  
  ]  
};
```



# Additional Transforms

- From `javafx.scene.transform`, you can also apply:
  - Affine, Scale, Shear and Translate transforms

# Effects: Drop Shadow Text

```
var text = Text {  
  content: "JavaSIG"  
  font: Font.font("Garmond",  
                  FontWeight.BOLD, 90)  
  x: 10,  
  y: 30,  
  textOrigin: TextOrigin.TOP  
  stroke: Color.BLACK  
  strokeWidth: 3  
  fill: Color.DEEP_SKY_BLUE  
}
```

The text "JavaSIG" is displayed in a blue, serif font. It is enclosed within a dashed orange rectangular border, which highlights the text without the drop shadow effect.The text "JavaSIG" is displayed in a blue, serif font. It has a drop shadow effect, where the text is rendered with a semi-transparent blue shadow offset downwards and to the right, giving it a 3D appearance.

# Effects: Drop Shadow

```
var shadowText = Text {
  content: "JavaSIG"
  font: Font.font("Garmond",
                  FontWeight.BOLD, 90)
  x: 10,
  y: 200,
  textOrigin: TextOrigin.TOP
  stroke: Color.BLACK
  strokeWidth: 3
  fill: Color.DEEP_SKY_BLUE
  effect:
    DropShadow {
      offsetX: 5
      offsetY: 15
      color: Color.GRAY
    }
}
```

JavaSIG

JavaSIG

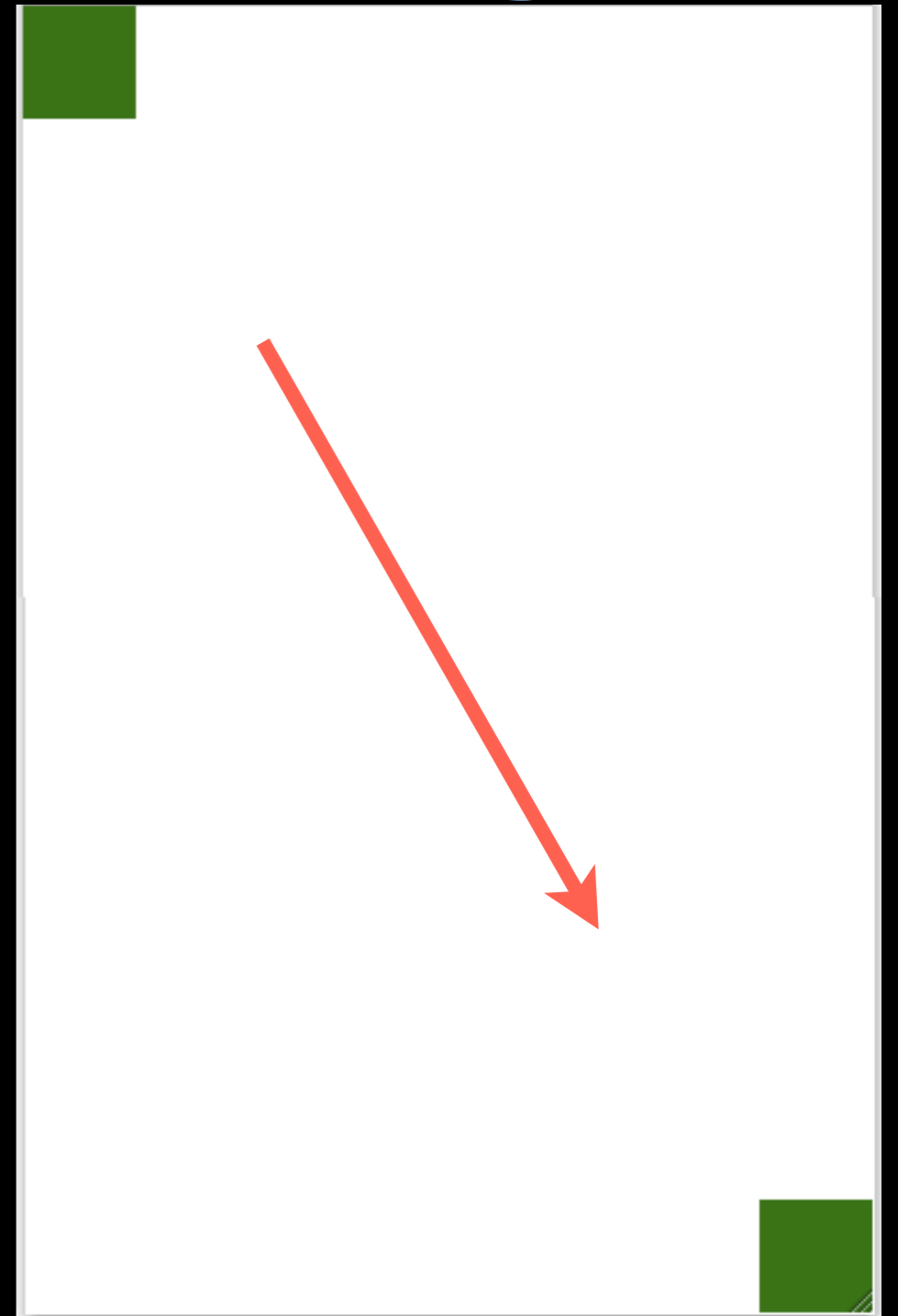
# Other Effects

- From `javafx.scene.effect`, you can also apply:
  - Blend, Flood, Glow, InnerShadow, Lighting, Motion Blur, and more!

# Animation

# Animation: Move A Rectangle

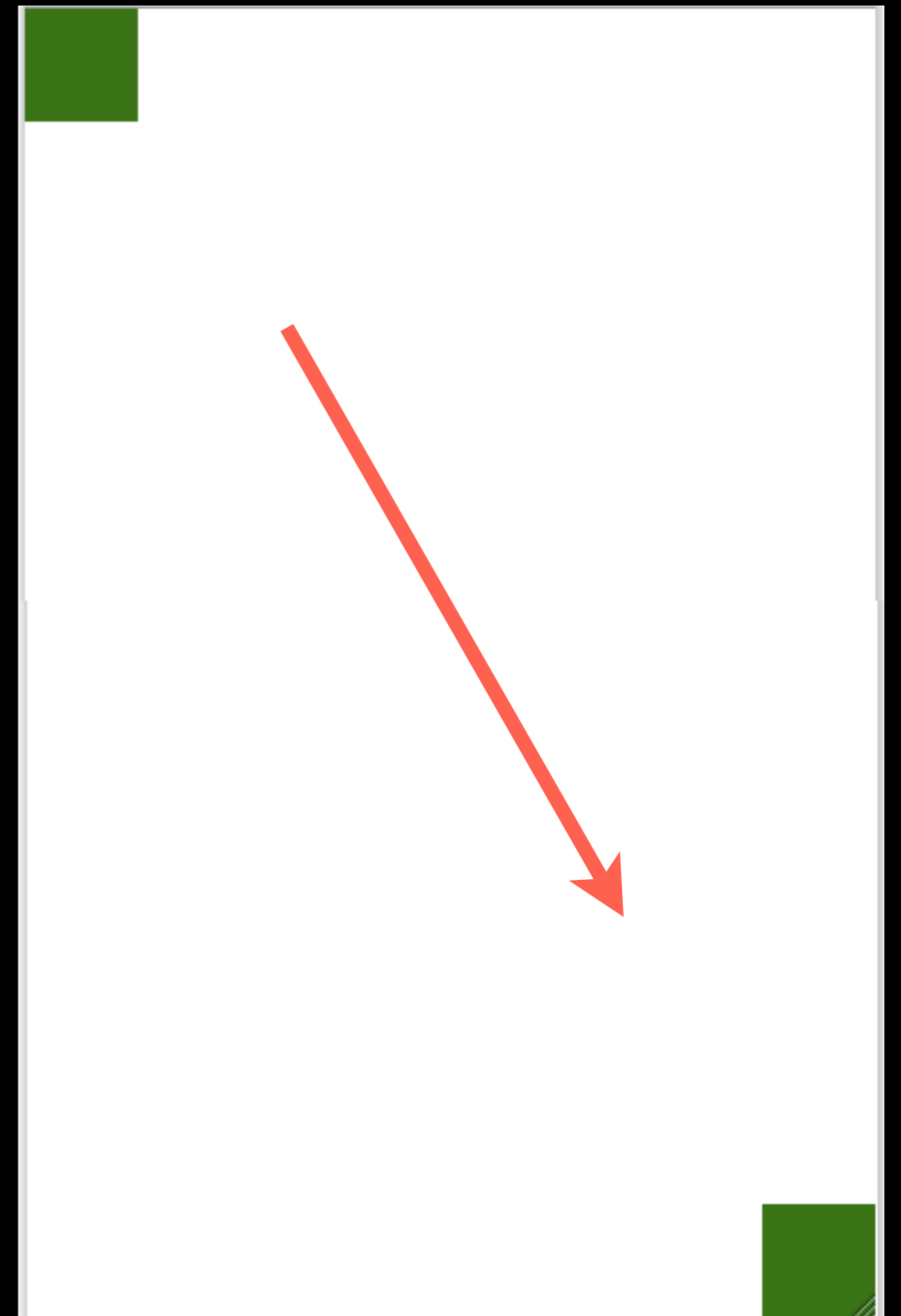
```
var xPosition = 0;  
var yPosition = 0;  
var rect = Rectangle {  
  x: bind xPosition  
  y: bind yPosition  
  height: 50  
  width: 50  
  fill: Color.DARKGREEN  
};
```



# Animation: Key Frame

```
var timeline = Timeline {
  repeatCount: 1
  keyFrames: [
    KeyFrame {
      time: 3s
      values: [
        xPosition => 325 tween
          Interpolator.LINEAR
        yPosition => 525 tween
          Interpolator.LINEAR
      ]
    }
  ]
};

// Starting the timeline
timeline.playFromStart();
```



# Animation: Tweening

`variable => endValue TWEEN Interpolator`

`xPosition => 325 TWEEN Interpolator.LINEAR`

`yPosition => 525 TWEEN Interpolator.LINEAR`



xPosition 0 increments evenly to 325

yPosition 0 increments evenly to 525

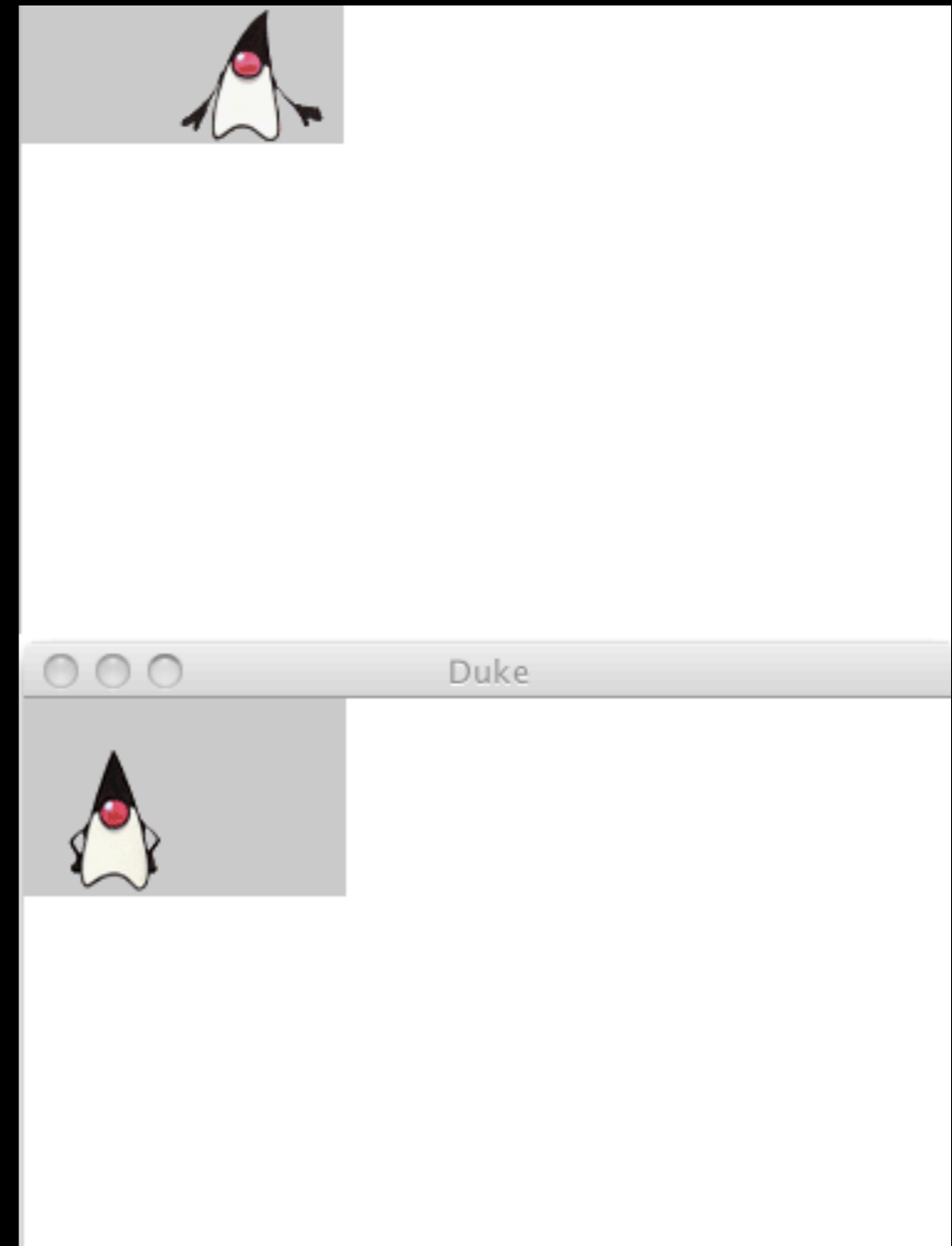
# Animation: Interpolation

- Interpolators: specify how to calculate current value of variable
- Built-in Interpolators:
  - LINEAR
  - EASEIN, EASEOUT, EASEBOTH
- Custom Interpolators implement Interpolatable

# Animation: Images

```
// we put Duke images T0-T16.gif the  
// duke directory, directly below this  
// directory
```

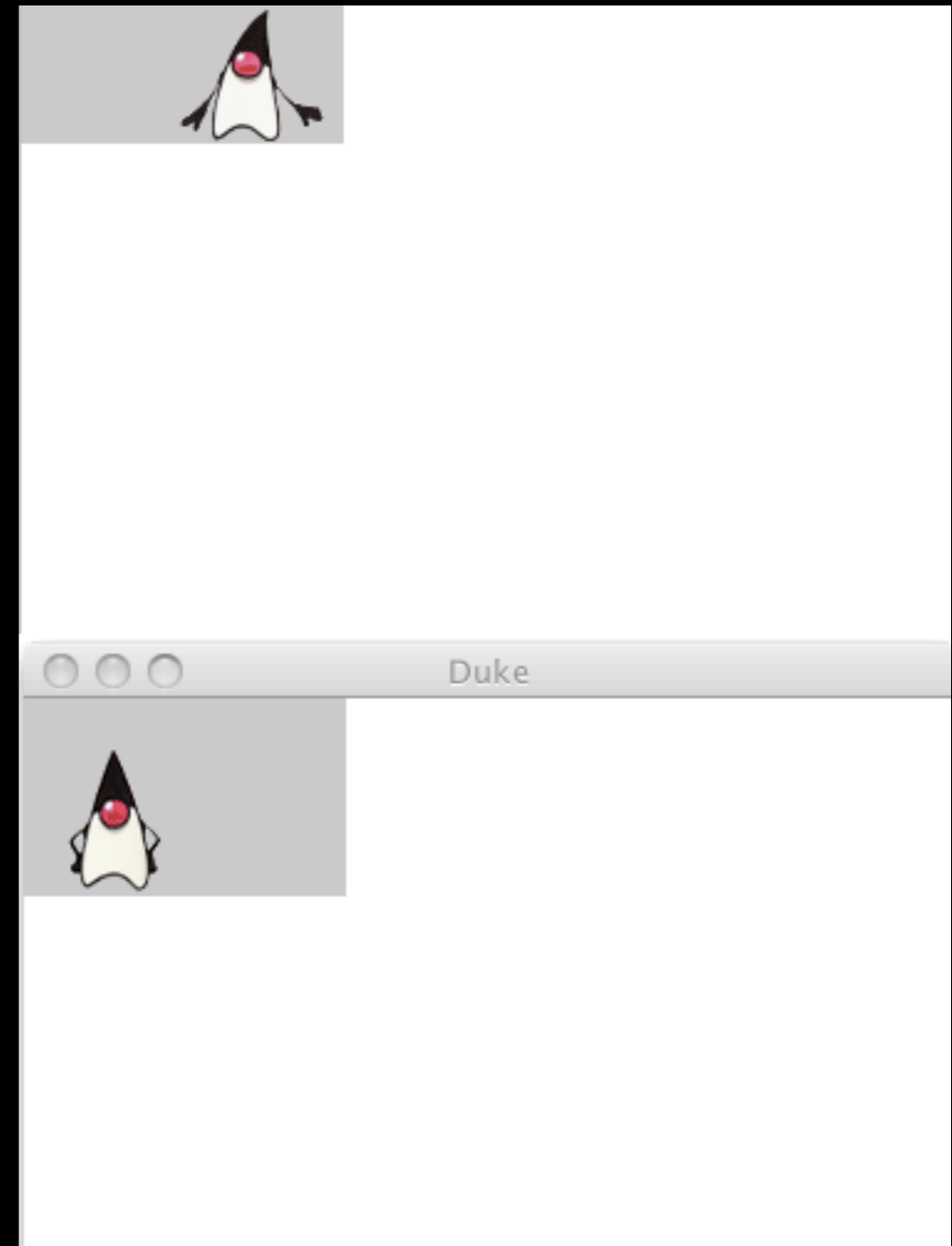
```
var frame = 0;  
var duke = ImageView {  
    image: bind Image {  
        url:  
            "{__DIR__}duke/T{frame}.gif";  
    }  
};
```



# Animation: Images

```
var timeline = Timeline {
    repeatCount: Timeline.INDEFINITE
    autoReverse: true
    keyFrames: [
        KeyFrame {
            time: 3s
            values: [
                frame => 16 tween
                Interpolator.LINEAR
            ]
        }
    ]
};

timeline.playFromStart();
```



# Events

# Events: Mouse

```
var message: String = "";  
var text = Text {  
    x: 150, y: 150, content: bind "{message}";  
}  
  
var rect = Rectangle {  
    x: 150, y: 10, width: 100, height: 100  
    fill: Color.ROYALBLUE  
    onMouseMoved: function( e: MouseEvent ):Void {  
        message = "mouse moved!"  
    }  
    onMouseClicked: function( e: MouseEvent ):Void {  
        message = "mouse clicked!"  
    }  
    onMousePressed: function( e: MouseEvent ):Void {  
        message = "mouse pressed!"  
    }  
};
```

# Events: Mouse

- Other Mouse Events (variables on Nodes):
  - onMouseEntered
  - onMouseReleased
  - onMouseDragged
  - onMouseWheelMoved
- All accept the same anonymous function

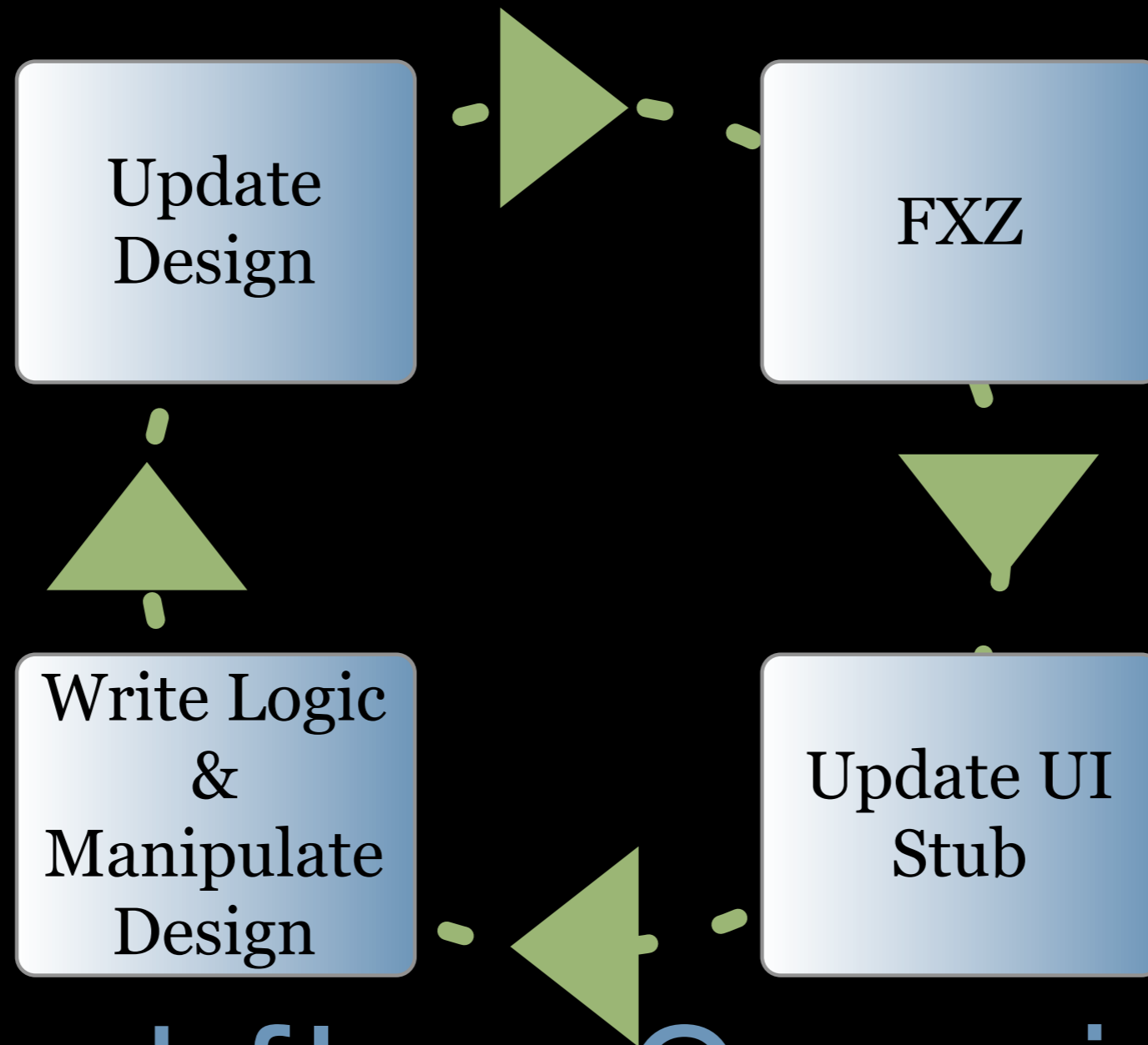
# Events: Keys

- Similar to Mouse events
- 3 types:
  - `onKeyPressed`, `onKeyReleased`,  
`onKeyTyped`
- Accept a function with this signature
  - `function( e: KeyEvent ):Void`

# Production Suite

# Production Suite

- Tools to *greatly* simplify designer/developer workflow
- Designers work in Illustrator, Photoshop or SVG capable tool
- Suite converts between (AI, PS and SVG) to JavaFX (FXZ)



# Workflow Overview



# Pig: Dice Game

# Demo of Game



# Demo of Artwork

# Design View

JavaFX Code!

```
ardAdapter.fx x Main.fx x Game.fx x Button.fx x
Archive [Icons]
points: [1425.65, 485.
fill: Color.rgb(0x26,
stroke: null
opacity: 0.5
},
},
Group {
id: "Dice_display"
content: [
Group {
content: [
Polygon {
points: [1178
fill: Color.W
stroke: null
},
SVGPath {
fill: Color.r
stroke: null
content: "M11
},
SVGPath {
fill: Color.r
```



# NetBeans View

# Rolling Dice

```
var die = fxz.getNode("die");  
var rotateDie = RotateTransition {  
    duration: 150ms  
    byAngle: 36  
    fromAngle: 0  
    toAngle: 360  
    repeatCount: 4  
    node: die  
}
```

Refers to Node in FXZ



```
rotateDie.playFromStart();
```

# Flashing Scores

```
public var value : Integer = -1 on replace {  
  if (value != -1) {  
    text.content = getContent(value);  
    var t = ScaleTransition {  
      repeatCount: 2  
      autoReverse: true  
      node: text  
      fromX: 1, toX: 1.3, byX: .05  
      fromY: 1, toY: 1.3, byY: .05  
      interpolate: Interpolator.EASEIN  
      duration: 175ms  
    }  
    t.playFromStart();  
  }  
}
```

Text node we'll grow



Grows to 1.3 x original size

# Making Noses Wiggle

```
var bigNoseWiggle = SequentialTransition {  
  repeatCount: Timeline.INDEFINITE  
  node: board.big_pig_nose;  
  content: [  
    PauseTransition {  
      duration: 8s  
    }  
    TranslateTransition {  
      repeatCount: 4  
      autoReverse: true  
      toY: -3, byY: 1  
      toX: -3, byX: 1  
      duration: 300ms  
      node: board.big_pig_nose;  
    }  
  ]  
};  
bigNoseWiggle.playFromStart();
```

One  
after  
the  
other

Refers to Node in FXZ

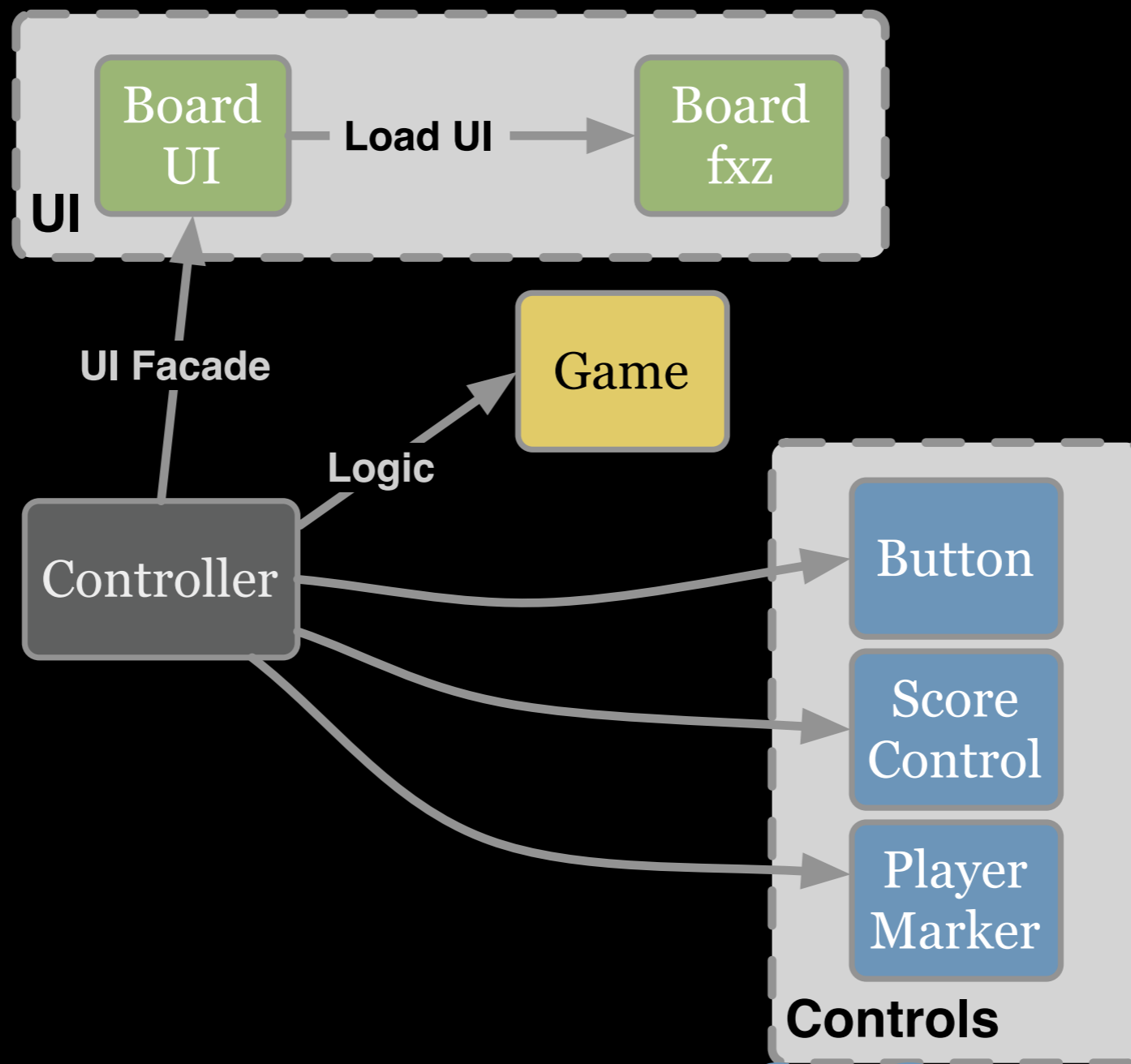
# Swapping Players

```
var shrink = Scale { x: .8, y: .8; }  
/** True if the current player is player1, false otherwise. */  
public var isPlayerOne: Boolean on replace {  
    if (isPlayerOne) {  
        player1Spot.visible = true;  
        player2Spot.visible = false;  
        swap(player1Label, player2Label);  
    } else {  
        player1Spot.visible = false;  
        player2Spot.visible = true;  
        swap(player2Label, player1Label);  
    }  
}
```

**Swap visible tab**

**Shrinks & lighten inactive name**

```
/** Swap the tabs to show the next player. */  
function swap( node1:Node, node2:Node ) {  
    delete shrink from node1.transforms;  
    var bounds = node2.boundsInParent;  
    transform.pivotX = bounds.minX + bounds.width / 2;  
    transform.pivotY = bounds.minY + bounds.height / 2;  
    insert shrink into node2.transforms;  
    node1.opacity = 1;  
    node2.opacity = .6;
```



# Structure of Game

# Pig Highlights

- Most of the code is the logic, effects & animation
- Smart “Controls” manipulate using Transitions
- Updates to UI through binds & triggers
- DIDN'T HAVE DRAW A THING!

# Production Suite: Pros

- Cleanly separates designer/developer roles
- Designers use tools they already know
- Developers manipulate design using standard JavaFX
  - (everything you just saw!)
- Teams can automate conversion using ANT

# Thank You!

- Special thanks to Jonathan Wei from Schematic
- For examples & source visit:
- <http://newfoo.net/presentations/introduction-to-javafx/>