Stagebot Technical Documentation

Rachael Thompson

This is documentation for a project I completed over the summer - building a robot for moving set pieces onstage. I designed and built it at home for use in my school's theatre. The robot is powered by an Arduino Uno, communicating with an Android app. Tactile control is important for stage equipment, so I modified a version of App Inventor to support input from a Gamepad through the OTG port, and used it to build a robot controlling app. The final product can carry 200 lbs, has a range of more than 400 feet, and is useful in creating a wide range of effects. Come Spring, I'm planning to rent it out to local school theaters.

You can see a video of it in action here: http://tinyurl.com/Stagebot
And you can find the details of my App Inventor modifications here: https://github.com/mit-cml/appinventor-sources/pull/843

Table of Contents:

Contact: rachaelst@mac.com

Purpose: To design a robot for use in a stage show, capable of carrying set pieces weighing between 50 and 150 lbs, for use under a standard 2'x2' platform.

Overview:

Parameter	Measurement	Unit
Height	7.5	inches
Width	21	inches
Length	21	inches
Weight	15	lbs
Powered Wheel Diameter	5	inches
Unpowered Wheel Diameter	1.5	inches
Unpowered Wheel Caster Height	1.75	inches
Height with Platform	9	inches
Width with Platform	24	inches
Length with Platform	24	inches
Weight of Platform	20	lbs
Carrying Capacity	200	lbs
Bluetooth Range	400	feet

Electrical Characteristics:

Parameter	Max	Average	Unit
Supply Voltage (DC)	12	12	Volt
Supply Current	50 (without fuse)	6	Amp
Battery Life	7.2		Amp Hours

Motor Specs:

Parameter	Stall	No Load	Unit
Supply Voltage	12	12	Volt
Supply Current	25	1.5	Amp
Speed	0	190	RPM

Construction - Bill of Materials:

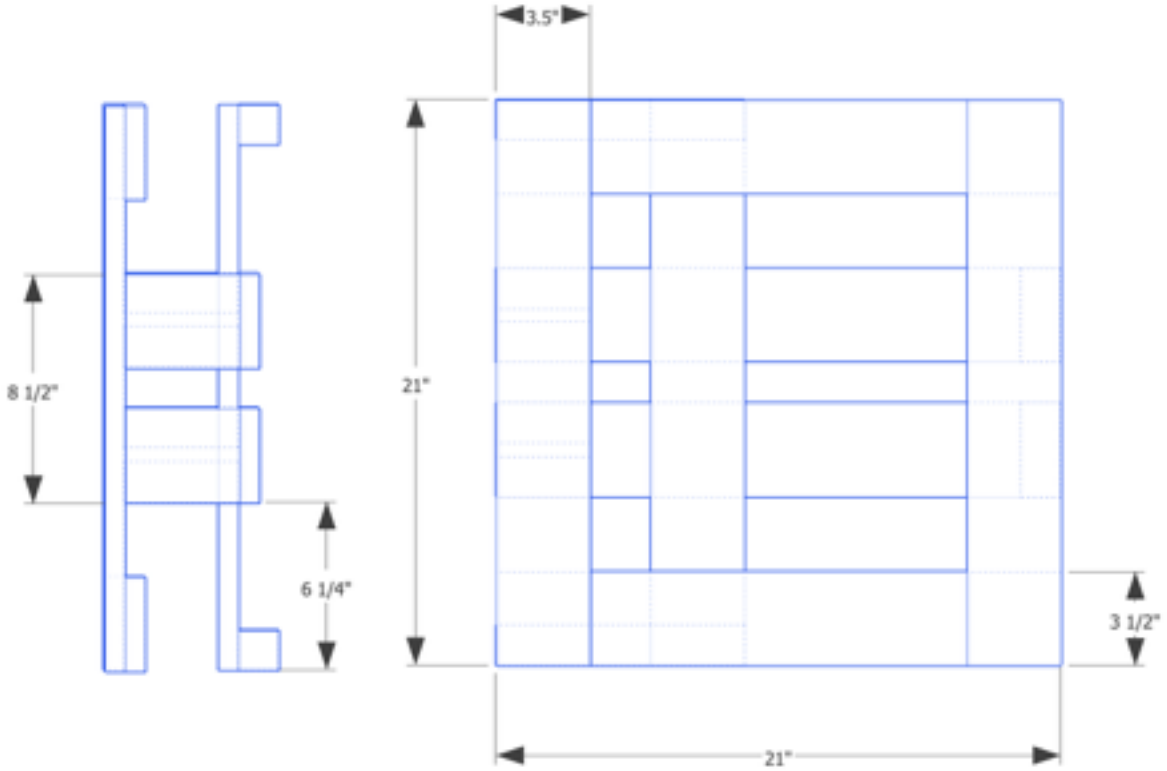
Stagebot Frame:

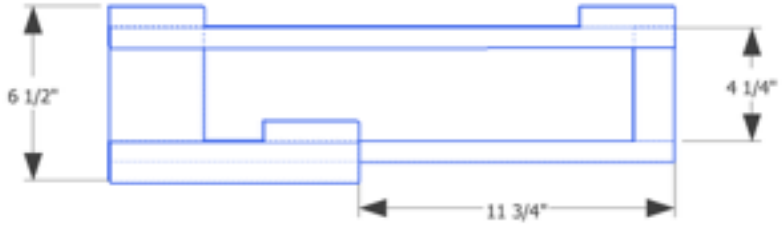
Material and Dimension	Amount
21” 1x4, White Pine	7
4.25” 2x4, White Pine	6
9.25” 2x2, White Pine	2
3” L-Braces	2
1 5/8” Drywall screws	20-25
3/4” Drywall Screws	8

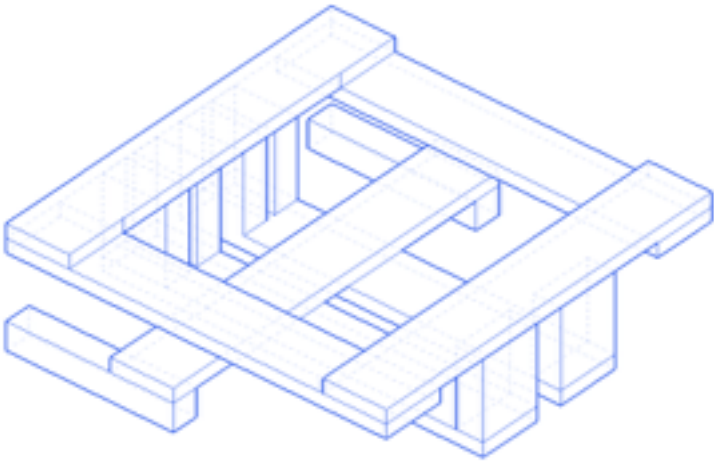
Platform:

Material and Dimension	Amount
2’x2’ Square of 3/4” plywood	1
24” 2x4, White Pine	2
21” 2x4, White Pine	2
9”x24” 1/4” Luan	4
3” Drywall Screws	10-15
1 5/8” Drywall Screws	8

Stagebot Frame Schematic:

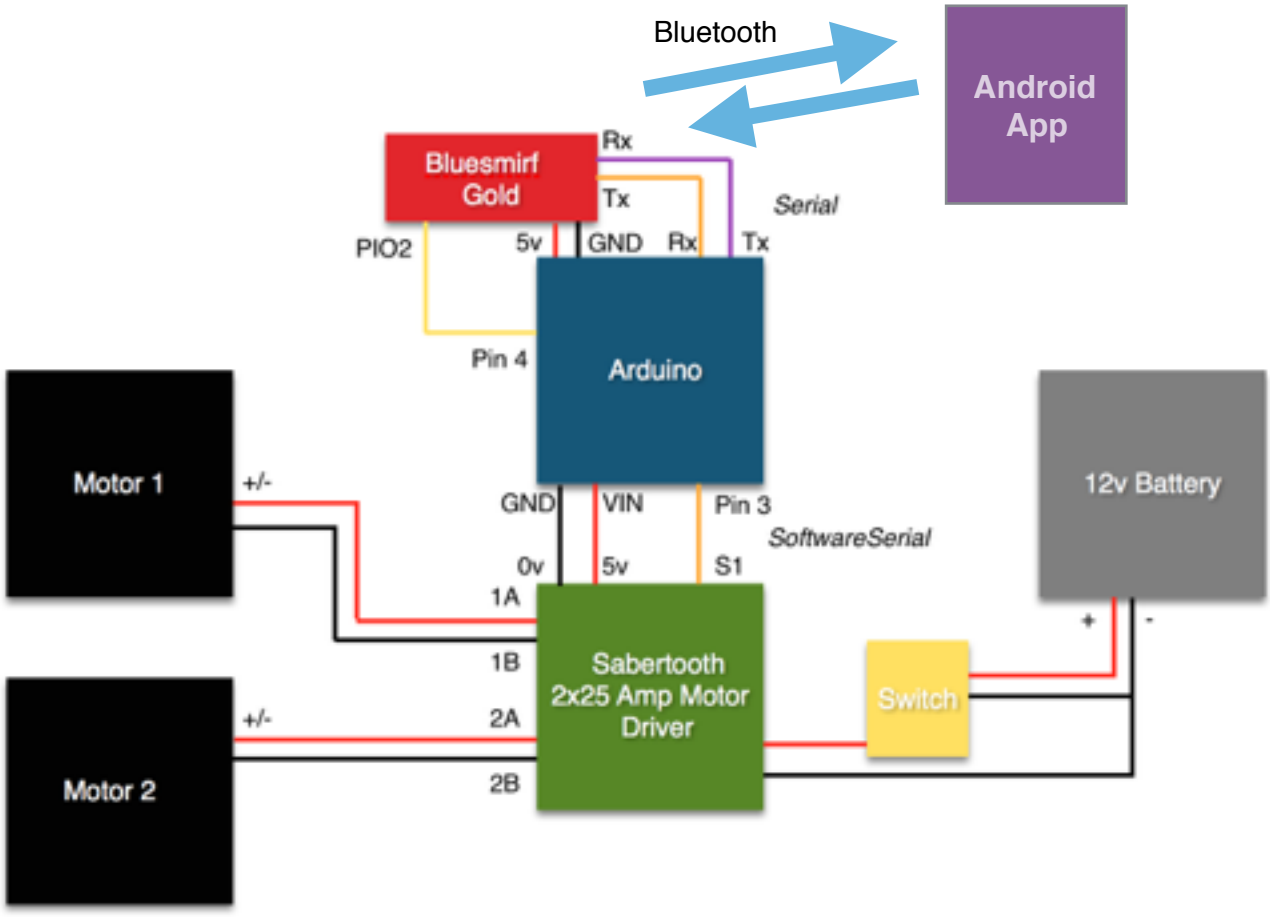






Name	Rachael Thompson
Part	Stagebot Frame
Material	#2 White Pine
Scale	NTS

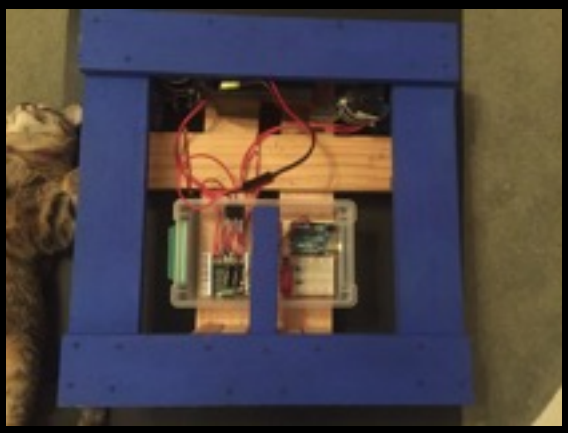
Wiring Diagram:



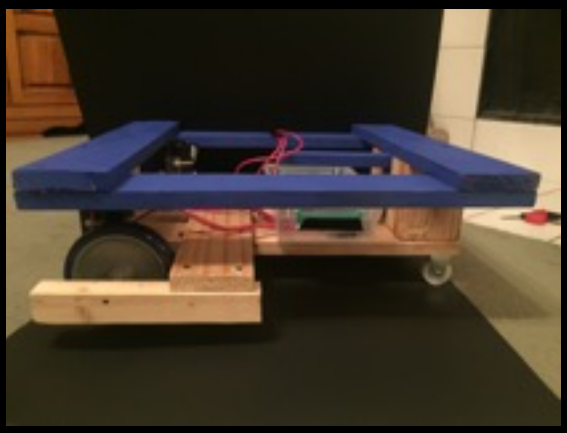
Component	Model
Microcontroller	Arduino Uno Rev3
Motor Driver	Sabertooth 2X25 V2
Bluetooth Chip	SparkFun Bluetooth Modem - BlueSMiRF Gold WRL-12582
Motor	190 Rpm 12Vdc Right Angle Drive Electric Motor
Battery	12v, 7.2 ah
Phone	Android ZTE Speed, running modified version of App Inventor 2

Photo Details:

Top View, with Cat



Side View



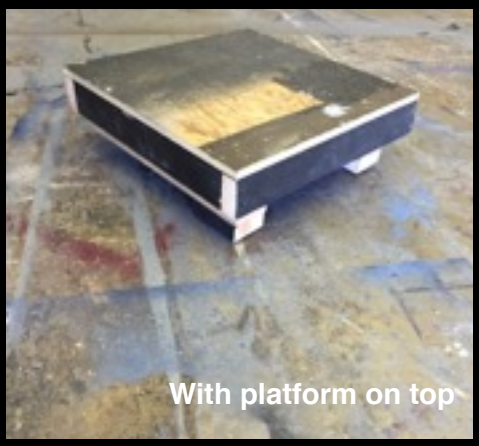
With box closed

Underside

Motor Driver

Arduino Wiring Detail

Motor mount detail



With platform on top

Code - SBGamepad for Arduino:

```
1   // This program receives and parses commands from an android device
    through bluetooth
2   // to control the motion of the Stagebot
3   // Author: Rachael T
4
5   #include <SoftwareSerial.h>
6
7   int maxSeconds = 10; // send status message every maxSeconds
8
9   int txPin = 3; // LED connected to pin 2 (on-board LED)
10  int rxPin =  2;
11  int connectedPin = 4; //Reads high when bluetooth is connected
12  int statusLED = 13;
13
14  volatile int seconds = 0;
15  volatile int lmotor;
16  volatile int rmotor;
17
18  String inputString = "";
19  String command = "";
20  String value = "";
21  boolean stringComplete = false; //Verifies that all bytes have been
    received
22
23  volatile int lSpeed;
24  volatile int rSpeed;
25
26  SoftwareSerial SaberSerial = SoftwareSerial( rxPin, txPin );
27
28  void setup(){
29    //Start serial connection
30    pinMode(txPin, OUTPUT);
31    pinMode(statusLED, OUTPUT);
32    pinMode(connectedPin, INPUT);
33    SaberSerial.begin(9600);
34
35    Serial.begin(9600);
36    Serial.print("Starting...");
37
38    inputString.reserve(50);
39    command.reserve(50);
40    value.reserve(50);
41  }
42
43  void loop(){
44
45        int cnct = digitalRead(connectedPin);
46        if(cnct == LOW){
47           SaberSerial.write(byte(0));
48         }
49         //Send command to motor driver
50         SaberSerial.write(lSpeed);
51         SaberSerial.write(rSpeed);
52  }
```

```
53
54  // Runs whenever bytes are recieved
55  void serialEvent() {
56    while (Serial.available()) {
57
58      // get the new byte:
59      char inChar = (char)Serial.read();
60
61      if (inChar == '\n' || inChar == '\r') {
62        stringComplete = true;  //message is over
63        break;
64      }
65
66      // add it to the inputString:
67       inputString += inChar;
68    }
69
70    if (stringComplete) {
71
72     //command structure is "CMD{Lmotor}z{Rmotor}z"
73     if (inputString.startsWith("CMD") && inputString.endsWith("z")) {
74        String inputL = "";
75        int pos = 0;
76        for(int i = 3; i < inputString.length(); i++){
77          if (inputString.charAt(i) != 'z'){
78            inputL += inputString.charAt(i);
79          }
80          else{
81            pos = i;
82            break;
83          }
84        }
85
86        String inputR = "";
87        for (int i = pos + 1; i < inputString.length(); i++) {
88          if (inputString.charAt(i) != 'z'){
89            inputR += inputString.charAt(i);
90          }
91          else{
92            pos = i;
93            break;
94          }
95        }
96
97        lSpeed = inputL.toInt();
98        rSpeed = inputR.toInt();
99
100       inputString = "";
101       stringComplete = false;
102     }
103   }
104
105 }
```

App Inventor Modifications:

Not including modifications to OdeMessages.

GamePad Component:

```
1   // -*- mode: java; c-basic-offset: 2; -*-
2   // Copyright 2009-2011 Google, All Rights reserved
3   // Copyright 2011-2012 MIT, All rights reserved
4   // Released under the Apache License, Version 2.0
5   // http://www.apache.org/licenses/LICENSE-2.0
6   //Component Created by Rachael T
7
8   package com.google.appinventor.components.runtime;
9
10  import com.google.appinventor.components.common.ComponentCategory;
11  import com.google.appinventor.components.annotations.DesignerComponent;
12  import com.google.appinventor.components.annotations.DesignerProperty;
13  import com.google.appinventor.components.runtime.util.ErrorMessages;
14  import com.google.appinventor.components.annotations.PropertyCategory;
15  import com.google.appinventor.components.common.PropertyTypeConstants;
16  import com.google.appinventor.components.annotations.SimpleEvent;
17  import com.google.appinventor.components.annotations.SimpleFunction;
18  import com.google.appinventor.components.annotations.SimpleObject;
19  import com.google.appinventor.components.annotations.SimpleProperty;
20  import com.google.appinventor.components.annotations.UsesLibraries;
21  import com.google.appinventor.components.annotations.UsesPermissions;
22  import com.google.appinventor.components.common.YaVersion;
23
24  import android.app.Activity;
25  import android.content.ContentValues;
26  import android.os.Environment;
27  import android.widget.FrameLayout;
28  import android.provider.MediaStore;
29  import android.util.Log;
30  import android.net.Uri;
31
32  import android.view.InputDevice;
33  import android.view.KeyEvent;
34  import android.view.MotionEvent;
35  import android.view.View.OnGenericMotionListener;
36  import android.view.View.OnKeyListener;
37  import android.view.View;
38
39  /**
40   * This component allows App inventor to support gamepad control by
    grabbing KeyEvents before they can reach the form's handler.
41   */
42  @DesignerComponent(version = 1,
43     description = "A component that allows Gamepad Controller support
    for your app",
44     category = ComponentCategory.EXTENSION,
45     nonVisible = true,
46     iconName = "images/extension.png")
47  @SimpleObject
```

```
48  @UsesPermissions(permissionNames =
    "android.permission.WRITE_EXTERNAL_STORAGE,
    android.permission.READ_EXTERNAL_STORAGE")
49
50  public class GamePad extends AndroidNonvisibleComponent
51      implements Component, KeyEvent.Callback,
    android.view.View.OnGenericMotionListener {
52
53
54    protected final Form form;
55
56
57    // Keypad Input variables
58    public boolean a;
59    private boolean b;
60    private boolean x;
61    private boolean y;
62    private boolean start;
63    private boolean select;
64    private boolean right_bumper;
65    private boolean left_bumper;
66    private boolean dpad_up;
67    private boolean dpad_down;
68    private boolean dpad_right;
69    private boolean dpad_left;
70
71    // Motion event input variables
72    private float left_joystick_x;
73    private float left_joystick_y;
74    private float right_joystick_x;
75    private float right_joystick_y;
76    private float right_trigger;
77    private float left_trigger;
78
79    /**
80     * Creates a Gamepad component.
81     *
82     * @param container container, component will be placed in
83     */
84    public GamePad(ComponentContainer container) {
85      super(container.$form());
86      this.form = container.$form();
87    }
88
89    /*This constructor is for testing purposes only */
90    public GamePad(Form form){
91      super(form);
92      this.form = form;
93    }
94
95
96    /**
97      * This method sorts out keycodes to register key presses
98      */
99    @Override
100   public boolean onKeyDown(int keyCode, KeyEvent event) {
101     if ((event.getSource() & InputDevice.SOURCE_GAMEPAD) ==
    InputDevice.SOURCE_GAMEPAD) {
```

```
102         if (keyCode == KeyEvent.KEYCODE_BUTTON_A) {
103           a = true;
104         }
105         if (keyCode == KeyEvent.KEYCODE_BUTTON_B) {
106           b = true;
107         }
108         if (keyCode == KeyEvent.KEYCODE_BUTTON_X) {
109           x = true;
110         }
111         if (keyCode == KeyEvent.KEYCODE_BUTTON_Y) {
112           y = true;
113         }
114         if (keyCode == KeyEvent.KEYCODE_BUTTON_R1) {
115           right_bumper = true;
116         }
117         if (keyCode == KeyEvent.KEYCODE_BUTTON_L1) {
118           left_bumper = true;
119         }
120         if (keyCode == KeyEvent.KEYCODE_BUTTON_START) {
121           start = true;
122         }
123         if (keyCode == KeyEvent.KEYCODE_BUTTON_SELECT) {
124           select = true;
125         }
126         if (keyCode == KeyEvent.KEYCODE_DPAD_UP) {
127           dpad_up = true;
128         }
129         if (keyCode == KeyEvent.KEYCODE_DPAD_DOWN) {
130           dpad_down = true;
131         }
132         if (keyCode == KeyEvent.KEYCODE_DPAD_LEFT) {
133           dpad_left = true;
134         }
135         if (keyCode == KeyEvent.KEYCODE_DPAD_RIGHT) {
136           dpad_right = true;
137         }
138       }
139       //Report that the key press was handled
140       return true;
141     }
142
143     /**
144       * This method sorts out keycodes to register key releases.
145       */
146     @Override
147     public boolean onKeyUp(int keyCode, KeyEvent event){
148       if ((event.getSource() & InputDevice.SOURCE_GAMEPAD) ==
    InputDevice.SOURCE_GAMEPAD) {
149         if (keyCode == KeyEvent.KEYCODE_BUTTON_A) {
150           a = false;
151         }
152         if (keyCode == KeyEvent.KEYCODE_BUTTON_B) {
153           b = false;
154         }
155         if (keyCode == KeyEvent.KEYCODE_BUTTON_X) {
156           x = false;
157         }
158         if (keyCode == KeyEvent.KEYCODE_BUTTON_Y) {
```

```
159         y = false;
160       }
161       if (keyCode == KeyEvent.KEYCODE_BUTTON_R1) {
162         right_bumper = false;
163       }
164       if (keyCode == KeyEvent.KEYCODE_BUTTON_L1) {
165         left_bumper = false;
166       }
167       if (keyCode == KeyEvent.KEYCODE_BUTTON_START) {
168         start = false;
169       }
170       if (keyCode == KeyEvent.KEYCODE_BUTTON_SELECT) {
171         select = false;
172       }
173       if (keyCode == KeyEvent.KEYCODE_DPAD_UP) {
174         dpad_up = false;
175       }
176       if (keyCode == KeyEvent.KEYCODE_DPAD_DOWN) {
177         dpad_down = false;
178       }
179       if (keyCode == KeyEvent.KEYCODE_DPAD_LEFT) {
180         dpad_left = false;
181       }
182       if (keyCode == KeyEvent.KEYCODE_DPAD_RIGHT) {
183         dpad_right = false;
184       }
185     }
186     //Report that the key release was handled
187     return true;
188   }
189
190   //Necessary to implement the Keycode.Callback interface, but not used
    by the method.
191   @Override
192   public boolean onKeyLongPress(int keyCode, KeyEvent event){
193     return true;
194   }
195
196   @Override
197   public boolean onKeyMultiple(int keyCode, int count, KeyEvent event){
198     return true;
199   }
200
201   //This method dispatches motion events to be processed one at a time
202   @Override
203   public boolean onGenericMotion(View v, MotionEvent event) {
204     // Check that the event came from a game controller
205     if ((event.getSource() & InputDevice.SOURCE_JOYSTICK) ==
    InputDevice.SOURCE_JOYSTICK && event.getAction() ==
    MotionEvent.ACTION_MOVE) {
206       // Process all historical movement samples in the batch
207       final int historySize = event.getHistorySize();
208       // Process the movements starting from the earliest historical
    position in the batch
209       for (int i = 0; i < historySize; i++) {
210         // Process the event at historical position i
211         processJoystickInput(event, i);
212       }
```

```
213         // Process the current movement sample in the batch (position -1)
214         processJoystickInput(event, -1);
215       }
216       return true;
217     }
218
219 //This process motion events for the joystick and DPAD (some DPADs use
    keycodes, some use an axis)
220 public void processJoystickInput(MotionEvent event, int historyPos) {
221
222       left_joystick_x = event.getAxisValue(MotionEvent.AXIS_X);;
223       left_joystick_y = event.getAxisValue(MotionEvent.AXIS_Y);
224       right_joystick_x = event.getAxisValue(12);
225       right_joystick_y = event.getAxisValue(13);
226
227       right_trigger = event.getAxisValue(MotionEvent.AXIS_RZ);
228       left_trigger = event.getAxisValue(MotionEvent.AXIS_Z);
229
230       float xaxis = event.getAxisValue(MotionEvent.AXIS_HAT_X);
231       float yaxis = event.getAxisValue(MotionEvent.AXIS_HAT_Y);
232
233       if (Float.compare(xaxis, -1.0f) == 0) {
234           dpad_left = true;
235           dpad_right = false;
236       } else if (Float.compare(xaxis, 1.0f) == 0) {
237           dpad_right = true;
238           dpad_left = false;
239       } else {
240           dpad_right = false;
241           dpad_left = false;
242       }
243
244       if (Float.compare(yaxis, -1.0f) == 0) {
245           dpad_up = true;
246           dpad_down = false;
247       } else if (Float.compare(yaxis, 1.0f) == 0) {
248           dpad_down = true;
249           dpad_up = false;
250       } else {
251           dpad_up = false;
252           dpad_down = false;
253       }
254 }
255
256 /**
257    * Returns true if the A button is pressed on the gamepad
258    *
259    * @return {@code true} the A button is pressed {@code false} the A
    button is not pressed
260    */
261 @SimpleProperty (description = "Is the A button pressed?", category =
    PropertyCategory.BEHAVIOR)
262   public boolean AButton(){
263       return a;
264   }
265
266 /**
267    * Returns true if the B button is pressed on the gamepad
```

```
268     *
269     * @return {@code true} the B button is pressed {@code false} the B
    button is not pressed
270     */
271 @SimpleProperty (description = "Is the B button pressed?", category =
    PropertyCategory.BEHAVIOR)
272   public boolean BButton(){
273       return b;
274   }
275
276 /**
277     * Returns true if the X button is pressed on the gamepad
278     *
279     * @return {@code true} the X button is pressed {@code false} the X
    button is not pressed
280     */
281 @SimpleProperty (description = "Is the X button pressed?", category =
    PropertyCategory.BEHAVIOR)
282   public boolean XButton(){
283       return x;
284   }
285
286 /**
287     * Returns true if the Y button is pressed on the gamepad
288     *
289     * @return {@code true} the Y button is pressed {@code false} the Y
    button is not pressed
290     */
291 @SimpleProperty (description = "Is the Y button pressed?", category =
    PropertyCategory.BEHAVIOR)
292   public boolean YButton(){
293       return y;
294   }
295
296 /**
297     * Returns true if the Right Bumper is pressed on the gamepad
298     *
299     * @return {@code true} the Right Bumper is pressed {@code false} the
    Right Bumper is not pressed
300     */
301 @SimpleProperty (description = "Is the Right bumper pressed?", category
    = PropertyCategory.BEHAVIOR)
302   public boolean RightBumper(){
303     return right_bumper;
304   }
305
306 /**
307     * Returns true if the Left Bumper is pressed on the gamepad
308     *
309     * @return {@code true} the Left Bumper is pressed {@code false} the
    Left Bumper is not pressed
310     */
311 @SimpleProperty (description = "Is the Left bumper pressed?", category
    = PropertyCategory.BEHAVIOR)
312   public boolean LeftBumper(){
313       return left_bumper;
314   }
315
```

```
316 /**
317    * Returns true if the Start button is pressed on the gamepad
318    *
319    * @return {@code true} the Start button is pressed {@code false} the
    Start button is not pressed
320    */
321 @SimpleProperty (description = "Is the Start button pressed?", category
    = PropertyCategory.BEHAVIOR)
322   public boolean Start(){
323       return start;
324   }
325
326 /**
327    * Returns true if the Select button is pressed on the gamepad
328    *
329    * @return {@code true} the Select button is pressed {@code false}
    the Select button is not pressed
330    */
331 @SimpleProperty (description = "Is the Select button pressed?",
    category = PropertyCategory.BEHAVIOR)
332   public boolean Select(){
333       return select;
334   }
335
336 /**
337    * Returns true if the DPAD Up is pressed on the gamepad
338    *
339    * @return {@code true} the DPAD Up is pressed {@code false} the DPAD
    Up is not pressed
340    */
341 @SimpleProperty (description = "Is the DPAD pressed up?", category =
    PropertyCategory.BEHAVIOR)
342   public boolean DpadUp(){
343       return dpad_up;
344   }
345
346 /**
347    * Returns true if the DPAD Down is pressed on the gamepad
348    *
349    * @return {@code true} the DPAD Down is pressed {@code false} the
    DPAD Down is not pressed
350    */
351 @SimpleProperty (description = "Is the DPAD pressed down?", category =
    PropertyCategory.BEHAVIOR)
352   public boolean DpadDown(){
353       return dpad_down;
354   }
355
356 /**
357    * Returns true if the DPAD Left is pressed on the gamepad
358    *
359    * @return {@code true} the DPAD Left is pressed {@code false} the
    DPAD Left is not pressed
360    */
361 @SimpleProperty (description = "Is the DPAD pressed left?", category =
    PropertyCategory.BEHAVIOR)
362   public boolean DpadLeft(){
363       return dpad_left;
```

```
364   }
365
366 /**
367    * Returns true if the DPAD Right is pressed on the gamepad
368    *
369    * @return {@code true} the DPAD Right is pressed {@code false} the
    DPAD Right is not pressed
370    */
371 @SimpleProperty (description = "Is the DPAD pressed right?", category =
    PropertyCategory.BEHAVIOR)
372   public boolean DpadRight(){
373       return dpad_right;
374   }
375
376 /**
377    * Returns the position of the Left joystick X-axis on the gamepad
378    *
379    * @return the float value of its x-axis position between 1 and -1
380    */
381 @SimpleProperty (description = "The Left Joystick X value", category =
    PropertyCategory.BEHAVIOR)
382   public float LeftJoystickX(){
383       return left_joystick_x;
384   }
385
386 /**
387    * Returns the position of the Left joystick Y-axis on the gamepad
388    *
389    * @return the float value of its y-axis position between 1 and -1
390    */
391 @SimpleProperty (description = "The Left Joystick Y value", category =
    PropertyCategory.BEHAVIOR)
392   public float LeftJoystickY(){
393       return left_joystick_y;
394   }
395
396 /**
397    * Returns the position of the Right joystick X-axis on the gamepad
398    *
399    * @return the float value of its x-axis position between 1 and -1
400    */
401 @SimpleProperty (description = "The Right Joystick X value", category =
    PropertyCategory.BEHAVIOR)
402   public float RightJoystickX(){
403       return right_joystick_x;
404   }
405
406 /**
407    * Returns the position of the Right joystick Y-axis on the gamepad
408    *
409    * @return the float value of its y-axis position between 1 and -1
410    */
411 @SimpleProperty (description = "The Right Joystick Y value", category =
    PropertyCategory.BEHAVIOR)
412   public float RightJoystickY(){
413       return right_joystick_y;
414   }
415
```

```
416 /**
417    * Returns the position of the Right Trigger on its axis
418    *
419    * @return the float value of its axis position between 1 and -1
420    */
421 @SimpleProperty (description = "The Right Trigger value", category =
    PropertyCategory.BEHAVIOR)
422   public float RightTrigger(){
423       return right_trigger;
424   }
425
426 /**
427    * Returns the position of the Left Trigger on its axis
428    *
429    * @return the float value of its axis position between 1 and -1
430    */
431 @SimpleProperty (description = "The Left Trigger value", category =
    PropertyCategory.BEHAVIOR)
432   public float LeftTrigger(){
433       return left_trigger;
434   }
435
436 /**
437   *This function enables the gamepad, and must be called before
    keycodes can be grabbed.
438   */
439 @SimpleFunction
440 public void EnableGamepad(){
441   form.dontGrabKeyEventsForComponent(this);
442   form.dontGrabMotionEventsForComponent(this);
443 }
444
445 /**
446   *This function disables the gamepad, leaving the EventDispatcher to
    deal with any key presses
447   */
448 @SimpleFunction
449 public void DisableGamepad(){
450   form.dontGrabKeyEventsForComponent(null);
451   form.dontGrabMotionEventsForComponent(null);
452 }
453
454
455
456 }
```

Modifications to Form.java:

```
1     //Contains only modifications by Rachael T
2
3     @DesignerComponent(version = YaVersion.FORM_COMPONENT_VERSION,
4       category = ComponentCategory.LAYOUT,
5       description = "Top-level component containing all other components
    in the program",
6       showOnPalette = false)
7   @SimpleObject
8   @UsesPermissions(permissionNames =
    "android.permission.INTERNET,android.permission.ACCESS_WIFI_STATE,andro
    id.permission.ACCESS_NETWORK_STATE")
9   public class Form extends Activity
10    implements Component, ComponentContainer, HandlesEventDispatching,
11    OnGlobalLayoutListener {
12
13    ...
14
15    public KeyEvent.Callback keyCatchingComponent;
16    public OnGenericMotionListener motionCatchingComponent;
17
18    ...
19
20    /*
21     * Here we override the hardware back button, just to make sure
22     * that the closing screen animation is applied. (In API level
23     * 5, we can simply override the onBackPressed method rather
24     * than bothering with onKeyDown)
25     *
26     * This method also redirects KeyEvents to components that use an OTG
    input device
27     */
28
29    @Override
30    public boolean onKeyDown(int keyCode, KeyEvent event) {
31      boolean handled = false;
32
33        if (keyCode == KeyEvent.KEYCODE_BACK) {
34          if (!BackPressed()) {
35            handled = super.onKeyDown(keyCode, event);
36            AnimationUtil.ApplyCloseScreenAnimation(this, closeAnimType);
37            return handled;
38         } else {
39           return true;
40         }
41        }
42
43    //This allows an OTG component to grab and use the KeyEvents
44     if(keyCatchingComponent != null){
45
46        try{
47          Class[] keyArgs = new Class[2];
48          keyArgs[0] = int.class;
49          keyArgs[1] = KeyEvent.class;
50
```

```
51          Method catchKey =
    keyCatchingComponent.getClass().getMethod("onKeyDown", keyArgs);
52          return ((Boolean)catchKey.invoke(keyCatchingComponent, keyCode,
    event));
53        } catch (Throwable e){
54          Log.e("Exception", e.getMessage());
55          System.out.println(e.getMessage());
56        }
57
58        return false;
59      }
60      else{
61        return super.onKeyDown(keyCode, event);
62      }
63    }
64
65    //For the use of components implementing OTG devices
66    @Override
67    public boolean onKeyUp(int keyCode, KeyEvent event){
68     if(keyCatchingComponent != null){
69        try{
70          Class[] keyArgs = new Class[2];
71          keyArgs[0] = int.class;
72          keyArgs[1] = KeyEvent.class;
73
74          Method catchKey =
    keyCatchingComponent.getClass().getMethod("onKeyUp", keyArgs);
75          return ((Boolean)catchKey.invoke(keyCatchingComponent, keyCode,
    event));
76        } catch (Throwable e){
77          Log.e("Exception", e.getMessage());
78        }
79
80        return false;
81      }
82      else{
83        return super.onKeyDown(keyCode, event);
84      }
85    }
86
87
88   //For the use of components implementing OTG devices
89    @Override
90    public boolean onGenericMotionEvent(MotionEvent event){
91      Log.d("FORM RT", event.toString());
92      if(motionCatchingComponent != null){
93        try{
94          Class[] motionArgs = new Class[2];
95          motionArgs[0] = View.class;
96          motionArgs[1] = MotionEvent.class;
97
98          Method catchMotion=
    motionCatchingComponent.getClass().getMethod("onGenericMotion",
    motionArgs);
99          return ((Boolean)catchMotion.invoke(motionCatchingComponent,
    getCurrentFocus(), event));
100       } catch (Throwable e){
101         Log.e("Exception", e.getMessage());
```

```
102         }
103         return false;
104       }
105       else{
106         return super.onGenericMotionEvent(event);
107       }
108     }
109
110
111     @SimpleEvent(description = "Device back button pressed.")
112     public boolean BackPressed() {
113       return EventDispatcher.dispatchEvent(this, "BackPressed");
114     }
115
116     ...
117
118 }
```

GamepadTest.java:

```
1   package com.google.appinventor.components.runtime;
2
3   import junit.framework.TestCase;
4   import android.view.KeyEvent;
5   import android.view.MotionEvent;
6   import android.view.View;
7
8   import org.easymock.EasyMock;
9   import org.junit.Before;
10  import org.junit.Test;
11  import org.junit.runner.RunWith;
12  import org.powermock.api.easymock.PowerMock;
13  import org.powermock.core.classloader.annotations.PrepareForTest;
14  import org.powermock.modules.junit4.PowerMockRunner;
15  import static org.junit.Assert.assertEquals;
16  import android.view.InputDevice;
17  /**
18   * Tests Gamepad.java
19   *
20   */
21  @RunWith(PowerMockRunner.class)
22  @PrepareForTest({KeyEvent.class, Form.class, MotionEvent.class})
23  public class GamePadTest {
24
25      private final Form formMock = PowerMock.createNiceMock(Form.class);
26      private final KeyEvent keyMock =
    PowerMock.createNiceMock(KeyEvent.class);
27      private final MotionEvent motionEventMock =
    PowerMock.createNiceMock(MotionEvent.class);
28      private GamePad gamepad= new GamePad(formMock);
29
30
31      @Before
32    public void setUp() throws Exception {
33      EasyMock.expect(keyMock.getSource()).andReturn(0x00000400 |
    0x00000001).anyTimes();
34      EasyMock.expect(motionEventMock.getSource()).andReturn(0x00000400 |
    0x00000001).anyTimes();
35
    EasyMock.expect(motionEventMock.getHistorySize()).andReturn(1).anyTimes
    ();
36    }
37
38      @Test
39      public void testA() throws Exception{
40          EasyMock.expect(keyMock.getKeyCode()).andReturn(96).anyTimes();
41          EasyMock.replay(keyMock);
42          gamepad.onKeyDown(keyMock.getKeyCode(), keyMock);
43          assertEquals(true, gamepad.AButton());
44          gamepad.onKeyUp(keyMock.getKeyCode(), keyMock);
45          assertEquals(false, gamepad.AButton());
46          EasyMock.reset(keyMock);
47      }
48      @Test
49      public void testB() throws Exception{
```

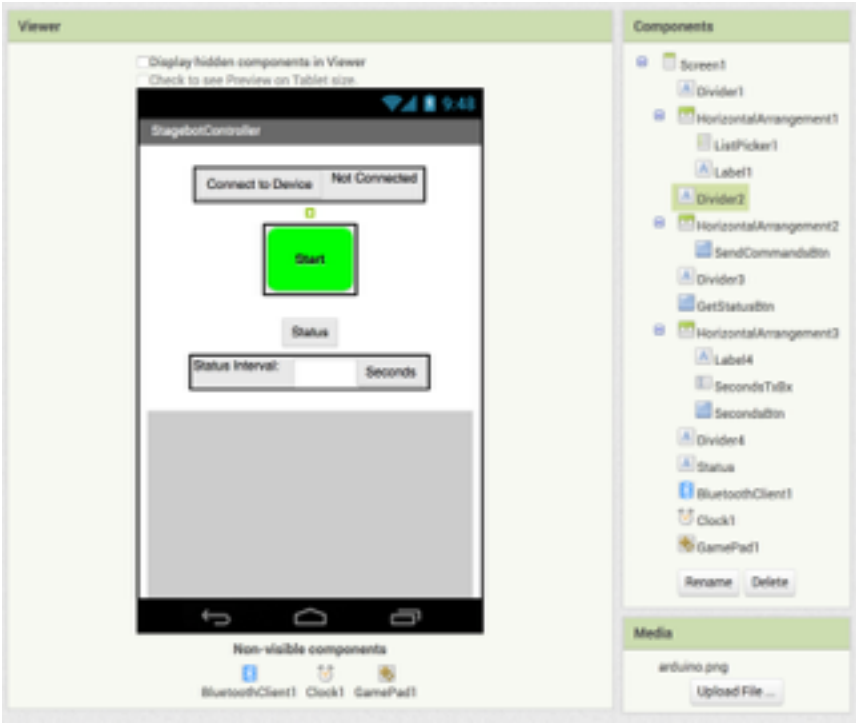
```
50          EasyMock.expect(keyMock.getKeyCode()).andReturn(97).anyTimes();
51          EasyMock.replay(keyMock);
52          gamepad.onKeyDown(keyMock.getKeyCode(), keyMock);
53          assertEquals(true, gamepad.BButton());
54          gamepad.onKeyUp(keyMock.getKeyCode(), keyMock);
55          assertEquals(false, gamepad.BButton());
56          EasyMock.reset(keyMock);
57      }
58      @Test
59      public void testX() throws Exception{
60          EasyMock.expect(keyMock.getKeyCode()).andReturn(99).anyTimes();
61          EasyMock.replay(keyMock);
62          gamepad.onKeyDown(keyMock.getKeyCode(), keyMock);
63          assertEquals(true, gamepad.XButton());
64          gamepad.onKeyUp(keyMock.getKeyCode(), keyMock);
65          assertEquals(false, gamepad.XButton());
66          EasyMock.reset(keyMock);
67      }
68      @Test
69      public void testY() throws Exception{
70
    EasyMock.expect(keyMock.getKeyCode()).andReturn(100).anyTimes();
71          EasyMock.replay(keyMock);
72          gamepad.onKeyDown(keyMock.getKeyCode(), keyMock);
73          assertEquals(true, gamepad.YButton());
74          gamepad.onKeyUp(keyMock.getKeyCode(), keyMock);
75          assertEquals(false, gamepad.YButton());
76          EasyMock.reset(keyMock);
77      }
78      @Test
79      public void testDpadUp() throws Exception{
80          EasyMock.expect(keyMock.getKeyCode()).andReturn(19).anyTimes();
81          EasyMock.replay(keyMock);
82          gamepad.onKeyDown(keyMock.getKeyCode(), keyMock);
83          assertEquals(true, gamepad.DpadUp());
84          gamepad.onKeyUp(keyMock.getKeyCode(), keyMock);
85          assertEquals(false, gamepad.DpadUp());
86          EasyMock.reset(keyMock);
87
88          Float q = new Float(-1);
89
    EasyMock.expect(motionEventMock.getAxisValue(MotionEvent.AXIS_HAT_Y)).a
    ndReturn(q).anyTimes();
90          EasyMock.replay(motionEventMock);
91          gamepad.processJoystickInput(motionEventMock, 0);
92          assertEquals(true, gamepad.DpadUp());
93          assertEquals(false, gamepad.DpadDown());
94          EasyMock.reset(motionEventMock);
95      }
96      @Test
97      public void testDpadDown() throws Exception{
98          EasyMock.expect(keyMock.getKeyCode()).andReturn(20).anyTimes();
99          EasyMock.replay(keyMock);
100         gamepad.onKeyDown(keyMock.getKeyCode(), keyMock);
101         assertEquals(true, gamepad.DpadDown());
102         gamepad.onKeyUp(keyMock.getKeyCode(), keyMock);
103         assertEquals(false, gamepad.DpadDown());
104         EasyMock.reset(keyMock);
```

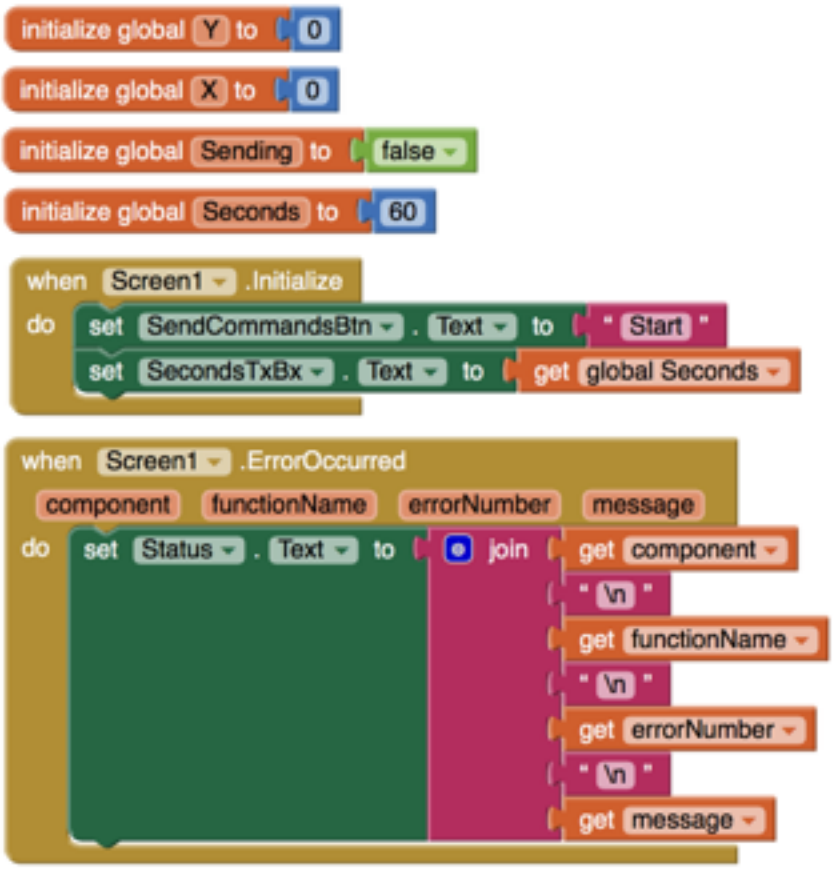
```
105
106         Float q = new Float(1);
107
    EasyMock.expect(motionEventMock.getAxisValue(MotionEvent.AXIS_HAT_Y)).a
    ndReturn(q).anyTimes();
108         EasyMock.replay(motionEventMock);
109         gamepad.processJoystickInput(motionEventMock, 0);
110         assertEquals(true, gamepad.DpadDown());
111         assertEquals(false, gamepad.DpadUp());
112         EasyMock.reset(motionEventMock);
113     }
114     @Test
115     public void testDpadRight() throws Exception{
116         EasyMock.expect(keyMock.getKeyCode()).andReturn(22).anyTimes();
117         EasyMock.replay(keyMock);
118         gamepad.onKeyDown(keyMock.getKeyCode(), keyMock);
119         assertEquals(true, gamepad.DpadRight());
120         gamepad.onKeyUp(keyMock.getKeyCode(), keyMock);
121         assertEquals(false, gamepad.DpadRight());
122         EasyMock.reset(keyMock);
123
124         Float q = new Float(1);
125
    EasyMock.expect(motionEventMock.getAxisValue(MotionEvent.AXIS_HAT_X)).a
    ndReturn(q).anyTimes();
126         EasyMock.replay(motionEventMock);
127         gamepad.processJoystickInput(motionEventMock, 0);
128         assertEquals(true, gamepad.DpadRight());
129         assertEquals(false, gamepad.DpadLeft());
130         EasyMock.reset(motionEventMock);
131     }
132     @Test
133     public void testDpadLeft() throws Exception{
134         EasyMock.expect(keyMock.getKeyCode()).andReturn(21).anyTimes();
135         EasyMock.replay(keyMock);
136         gamepad.onKeyDown(keyMock.getKeyCode(), keyMock);
137         assertEquals(true, gamepad.DpadLeft());
138         gamepad.onKeyUp(keyMock.getKeyCode(), keyMock);
139         assertEquals(false, gamepad.DpadLeft());
140         EasyMock.reset(keyMock);
141
142         Float q = new Float(-1);
143
    EasyMock.expect(motionEventMock.getAxisValue(MotionEvent.AXIS_HAT_X)).a
    ndReturn(q).anyTimes();
144         EasyMock.replay(motionEventMock);
145         gamepad.processJoystickInput(motionEventMock, 0);
146         assertEquals(true, gamepad.DpadLeft());
147         assertEquals(false, gamepad.DpadRight());
148         EasyMock.reset(motionEventMock);
149     }
150     @Test
151     public void testRightBumper() throws Exception{
152
    EasyMock.expect(keyMock.getKeyCode()).andReturn(103).anyTimes();
153         EasyMock.replay(keyMock);
154         gamepad.onKeyDown(keyMock.getKeyCode(), keyMock);
155         assertEquals(true, gamepad.RightBumper());
```

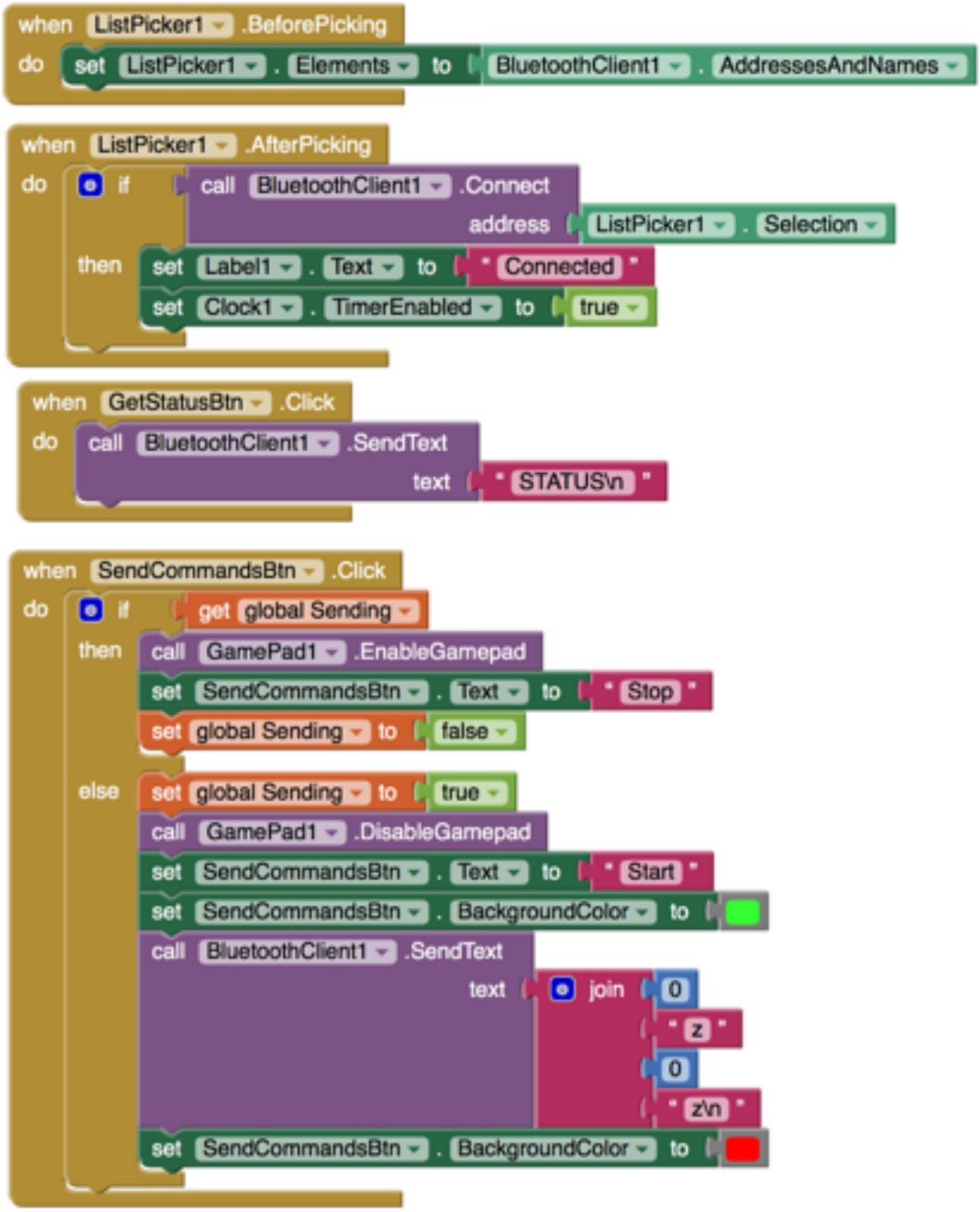
```
156         gamepad.onKeyUp(keyMock.getKeyCode(), keyMock);
157         assertEquals(false, gamepad.RightBumper());
158         EasyMock.reset(keyMock);
159 
160     }
161     @Test
162     public void testLeftBumper() throws Exception{
163 
    EasyMock.expect(keyMock.getKeyCode()).andReturn(102).anyTimes();
164         EasyMock.replay(keyMock);
165         gamepad.onKeyDown(keyMock.getKeyCode(), keyMock);
166         assertEquals(true, gamepad.LeftBumper());
167         gamepad.onKeyUp(keyMock.getKeyCode(), keyMock);
168         assertEquals(false, gamepad.LeftBumper());
169         EasyMock.reset(keyMock);
170     }
171     @Test
172     public void testLeftJoystickX() throws Exception{
173         Float q = new Float(1);
174 
    EasyMock.expect(motionEventMock.getAxisValue(MotionEvent.AXIS_X)).andRe
    turn(q).anyTimes();
175         EasyMock.replay(motionEventMock);
176         gamepad.processJoystickInput(motionEventMock, 0);
177         assertEquals(1, gamepad.LeftJoystickX(), 0);
178         EasyMock.reset(motionEventMock);
179         q = new Float(0);
180 
    EasyMock.expect(motionEventMock.getAxisValue(MotionEvent.AXIS_X)).andRe
    turn(q).anyTimes();
181         EasyMock.replay(motionEventMock);
182         gamepad.processJoystickInput(motionEventMock, 0);
183         assertEquals(0, gamepad.LeftJoystickX(), 0);
184     }
185     @Test
186     public void testLeftJoystickY() throws Exception{
187         Float q = new Float(1);
188 
    EasyMock.expect(motionEventMock.getAxisValue(MotionEvent.AXIS_Y)).andRe
    turn(q).anyTimes();
189         EasyMock.replay(motionEventMock);
190         gamepad.processJoystickInput(motionEventMock, 0);
191         assertEquals(1, gamepad.LeftJoystickY(), 0);
192         EasyMock.reset(motionEventMock);
193         q = new Float(0);
194 
    EasyMock.expect(motionEventMock.getAxisValue(MotionEvent.AXIS_Y)).andRe
    turn(q).anyTimes();
195         EasyMock.replay(motionEventMock);
196         gamepad.processJoystickInput(motionEventMock, 0);
197         assertEquals(0, gamepad.LeftJoystickY(), 0);
198     }
199     @Test
200     public void testRightJoystickX() throws Exception{
201         Float q = new Float(1);
202 
    EasyMock.expect(motionEventMock.getAxisValue(12)).andReturn(q).anyTimes
    ();
```

```
203         EasyMock.replay(motionEventMock);
204         gamepad.processJoystickInput(motionEventMock, 0);
205         assertEquals(1, gamepad.RightJoystickX(), 0);
206         EasyMock.reset(motionEventMock);
207         q = new Float(0);
208 
    EasyMock.expect(motionEventMock.getAxisValue(12)).andReturn(q).anyTimes
    ();
209         EasyMock.replay(motionEventMock);
210         gamepad.processJoystickInput(motionEventMock, 0);
211         assertEquals(0, gamepad.RightJoystickX(), 0);
212     }
213     @Test
214     public void testRightJoystickY() throws Exception{
215         Float q = new Float(1);
216 
    EasyMock.expect(motionEventMock.getAxisValue(13)).andReturn(q).anyTimes
    ();
217         EasyMock.replay(motionEventMock);
218         gamepad.processJoystickInput(motionEventMock, 0);
219         assertEquals(1, gamepad.RightJoystickY(), 0);
220         EasyMock.reset(motionEventMock);
221         q = new Float(0);
222 
    EasyMock.expect(motionEventMock.getAxisValue(13)).andReturn(q).anyTimes
    ();
223         EasyMock.replay(motionEventMock);
224         gamepad.processJoystickInput(motionEventMock, 0);
225         assertEquals(0, gamepad.RightJoystickY(), 0);
226     }
227 
228 }
```

App Code:


Viewer
Display hidden components in Viewer
Check to see Preview on Tablet size.
StagebotController
Connect to Device
Not Connected
0
Start
Status
Status Interval:
Seconds
Non-visible components
BluetoothClient1
Clock1
GamePad1
Components
Screen1
Divider1
HorizontalArrangement1
ListPicker1
Label1
Divider2
HorizontalArrangement2
SendCommandsBtn
Divider3
GetStatusBtn
HorizontalArrangement3
Label4
SecondsTxBx
SecondsBtn
Divider4
Status
BluetoothClient1
Clock1
GamePad1
Rename
Delete
Media
arduino.png
Upload File ...


initialize global Y to 0
initialize global X to 0
initialize global Sending to false
initialize global Seconds to 60
when Screen1 .Initialize
do set SendCommandsBtn . Text to " Start "
set SecondsTxBx . Text to get global Seconds
when Screen1 .ErrorOccurred
component functionName errorNumber message
do set Status . Text to join get component
" \n "
get functionName
" \n "
get errorNumber
" \n "
get message


when ListPicker1 .BeforePicking
do set ListPicker1 . Elements to BluetoothClient1 . AddressesAndNames
when ListPicker1 .AfterPicking
do if call BluetoothClient1 .Connect
address ListPicker1 . Selection
then set Label1 . Text to " Connected "
set Clock1 . TimerEnabled to true
when GetStatusBtn .Click
do call BluetoothClient1 .SendText
text " STATUS\n "
when SendCommandsBtn .Click
do if get global Sending
then call GamePad1 .EnableGamepad
set SendCommandsBtn . Text to " Stop "
set global Sending to false
else set global Sending to true
call GamePad1 .DisableGamepad
set SendCommandsBtn . Text to " Start "
set SendCommandsBtn . BackgroundColor to
call BluetoothClient1 .SendText
text join 0
" z "
0
" z\n "
set SendCommandsBtn . BackgroundColor to