



Abstract

Visual Programming Languages are valuable tools for teaching programming concepts, allowing visual abstraction of code blocks. Blockly is a web-based, client-side JavaScript library for rapidly building visual programming editors. We discuss the development of a visual programming application with Blockly, and present two case studies created for this project that demonstrate the power and flexibility of Blockly.

Introduction

A relatively new breed of programming environments has been growing to address the learning needs of those programming for the first time. These environments—such as MIT’s Scratch, Berkeley’s Snap!, or the open source Blockly—allow drag-and-drop “blocks” to be arranged into visually sequential programs. This approach prevents syntactical errors, which can be a significant hurdle to initial learners, and allows beginners to instead focus on the logic of programming without deep concern for syntax.

Integrating Blockly

Blockly is entirely client-side and must be included into the web page by both loading the necessary JavaScript files and by injecting Blockly into a fixed-size div or a resizable iframe. Once downloaded from [1], the scripts that need to be loaded are below:

```
<!--The Blockly Engine-->
<script src="blockly/blockly_compressed.js">
</script>
<!--Predefined Blocks-->
<script src="blockly/blocks_compressed.js">
</script>
<!--User Messages-->
<script src="blockly/msg/js/en.js"></script>
<!--Javascript generator for blockly-->
<script src="blockly/javascript_compressed.js">
</script>
```

After including the above script tags in a web app’s HTML, Blockly may be injected into an empty <div> element with JavaScript similar to the following:

```
<script type="text/javascript">
  Blockly.inject(document.getElementById(
    'blocklyDiv'), {path: '../',
    toolbox: document.getElementById('toolbox')});
</script>
```

Pixly: Media Computation

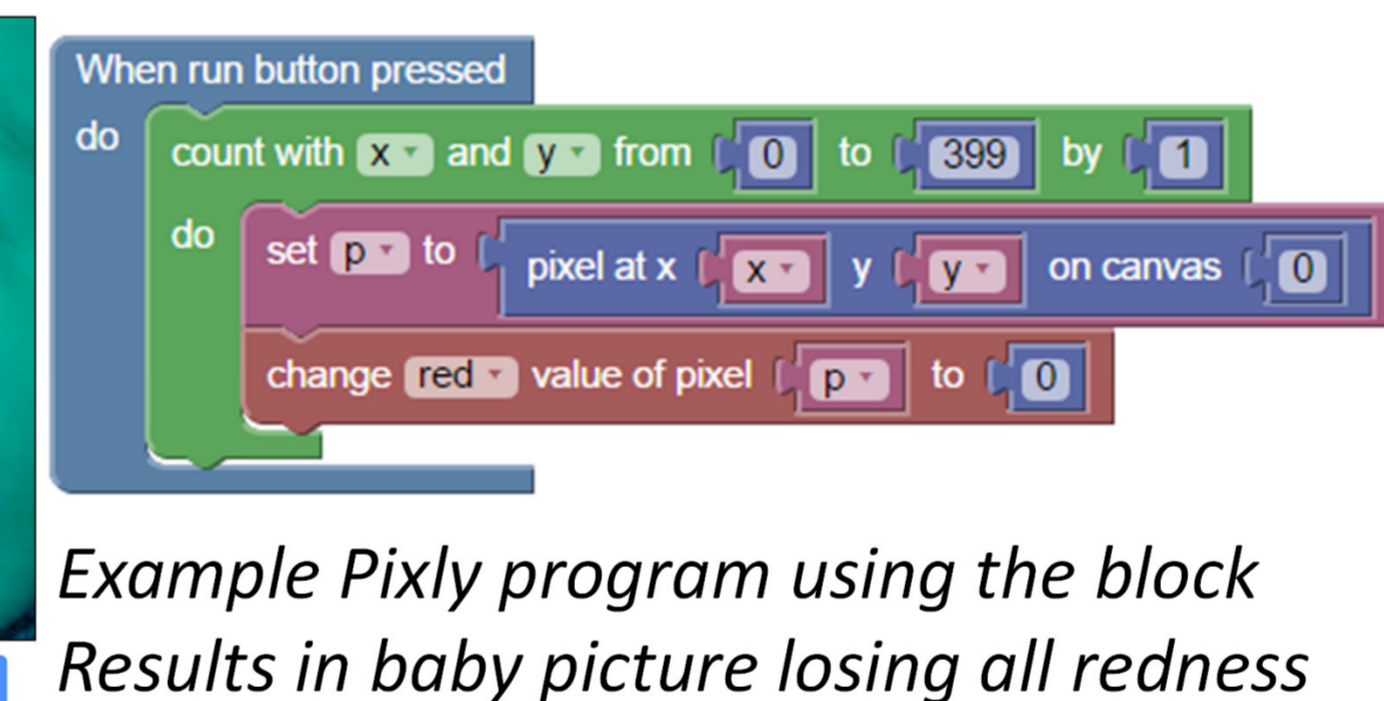
Pixly [2] provides an environment for students to practice media computation [3] (in particular, pixel manipulation of images). Using custom Blockly blocks, users can iterate through and manipulate the RGB values of the pixels of an image.

In order to make full use of the Blockly library, it is necessary to define custom blocks that interact with a web application’s internal API. The implementation of custom blocks requires both a definition—in which the Blockly.Blocks API is extended to contain the name, shape, color, label, and other attributes (e.g., variables)— and its corresponding code generation, which returns a string of code to be executed upon the block’s execution. Shown to the bottom-right is the definition and code generation for the “change red value of pixel p to 0.”

```
Blockly.Blocks['setPixelRGB'] = {
  init: function() {
    var RGB = [{"red", 'r'}, (...)];
    this.setColour(0);
    this.appendDummyInput()
      .appendField("change")
      .appendField(new
        Blockly.FieldDropdown(RGB), 'RGB')
      .appendField("value of pixel");
    this.appendValueInput("PIXEL")
      .setCheck("Number");
    this.appendValueInput("VAL")
      .setCheck("Number").appendField("to");
    this.setInputsInline(true);
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setTooltip('Set R, G, or B value.');
```

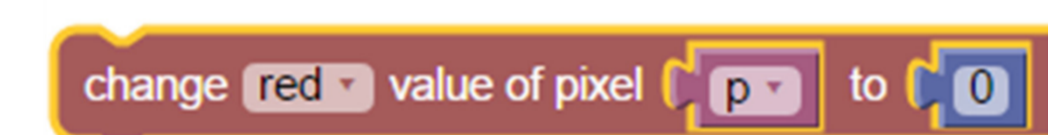
```
Blockly.JavaScript['setPixelRGB'] = function(block) {
  var ord = Blockly.JavaScript.ORDER_ASSIGNMENT;
  var rgb = block.getFieldValue('RGB');
  var p = Blockly.JavaScript.valueToCode(block, 'PIXEL', ord);
  var val = Blockly.JavaScript.valueToCode(block, 'VAL', ord);
  var code = "setPixelRGB("+p+", '"+rgb+"', '"+val+"');\n";
  return code;
};
```

Specify the JavaScript code returned by the Block



Example Pixly program using the block
Results in baby picture losing all redness

Specify the shape/color/text/inputs of Block



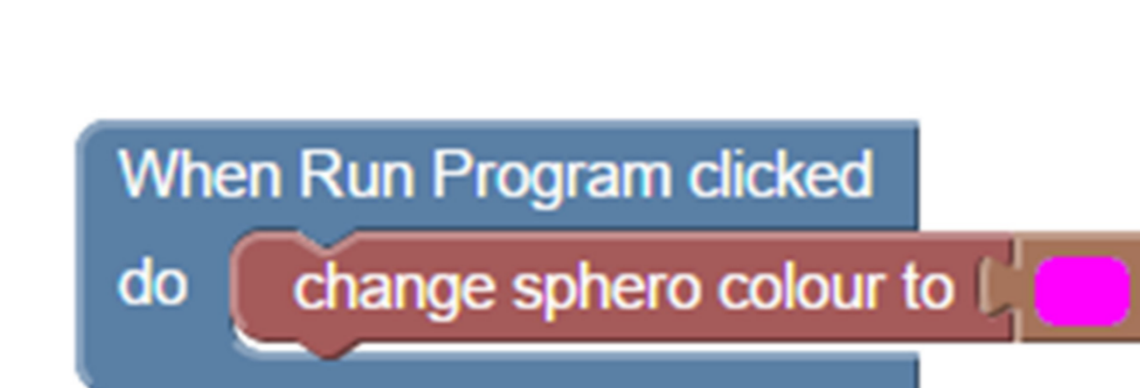
Blockly renders the block defined above

Spherly: Robotic Control

Spherly [3] communicates via a WebSocket with a local server application that uses Bluetooth to send commands and retrieve responses from the Sphero (a programmable robot ball). Additionally, a client-side command queue is used to allow the speed of command execution to be varied.

```
Sphero.setRGB = function(colour_rgb){
  Sphero.command_queue.push(["setRGB", colour_rgb]);
}
```

Function call (above) corresponding to block (below)



Spherly Color Change Code and Result on Sphero

```
Sphero.execute = function(){
  var command = this.command_queue.shift();
  if (command == undefined) return;
  switch(command[0]){
    case "setRGB":
      var color = command[1];
      var command = {"command": "setRGB",
        "red": color.r,
        "green": color.g,
        "blue": color.b};
      window.connection.send(command);
      /*The server handles the "setRGB" command
      by creating a packet described
      by the Sphero API [4]*/
      break;
    default: break;
  }
  setTimeout(Sphero.execute, 10);
}
```

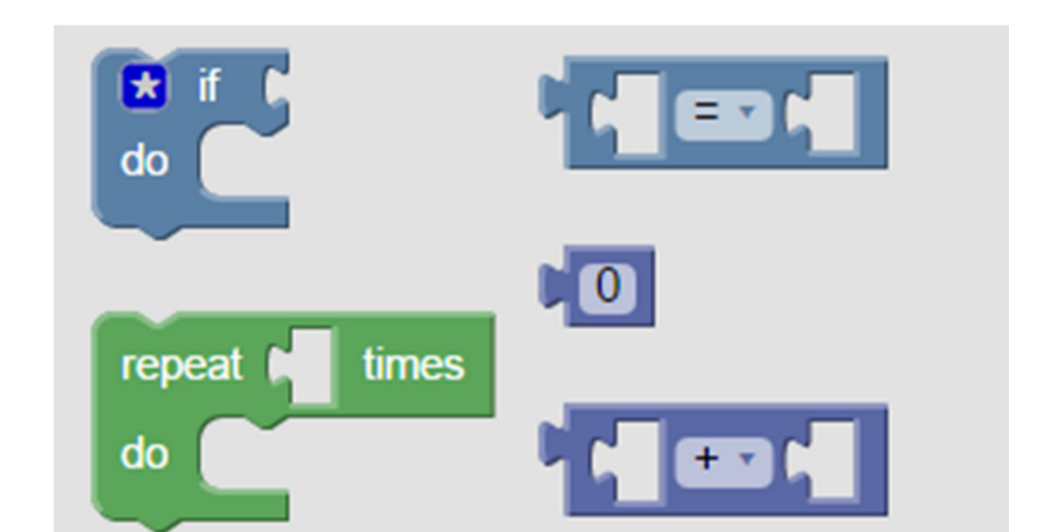
Spherly code to handle execution of a command queue

Including Blocks

Blocks must be included in an xml “toolbox” in the app’s HTML:

```
<xml id="toolbox" style="display: none">
  <block type="controls_if"></block>
  <block type="controls_repeat_ext"></block>
  <block type="logic_compare"></block>
  <block type="math_number"></block>
  <block type="math_arithmetic"></block>
</xml>
```

Blocks specified in the toolbox must either be predefined by Blockly or created by the user in the web application’s scripts.



Toolbox described by xml above

Conclusion

- Visual Programming Languages allow logical abstractions and hide syntax from learners
- Blockly allows developers to quickly, easily, and flexibly create new applications that take advantage of the power of Visual Programming Language
- Pixly and Spherly exemplify the range of possibilities afforded by Blockly
- Future work includes refining Pixly and Spherly, as well as additional Blockly applications for other contexts (e.g., JavaScript Sound Generation, Lego EV3 robots)

References

- [1] Blockly Installation Instructions:
<https://developers.google.com/blockly/installation/overview>
- [2] Pixly website:
<http://outreach.cs.ua.edu/pixly/>
- [3] Spherly website:
<http://outreach.cs.ua.edu/spherly/>
- [4] Sphero API:
<http://orbotixinc.github.io/Sphero-Docs/docs/sphero-api/packet-structures.html>

Acknowledgements



This work was sponsored in part by NSF awards #117940 and #1240944.