```
/**
 * Blockly Games: Bird Blocks
 *
 * Copyright 2013 Google Inc.
 * https://github.com/google/blockly-games
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/**
 * @fileoverview Blocks for Blockly's Bird application.
 * @author q.neutron@gmail.com (Quynh Neutron)
 */
'use strict';

goog.provide('Bird.Blocks');

goog.require('Blockly');
goog.require('Blockly.Blocks.logic');
goog.require('Blockly.Blocks.math');
goog.require('Blockly.JavaScript');
goog.require('Blockly.JavaScript.logic');
goog.require('Blockly.JavaScript.math');
goog.require('BlocklyGames');


/**
 * Common HSV hue for all variable blocks.
 */
Bird.Blocks.VARIABLES_HUE = 330;

/**
 * HSV hue for movement block.
 */
Bird.Blocks.MOVEMENT_HUE = 290;

// Extensions to Blockly's language and JavaScript generator.

Blockly.Blocks['bird_noWorm'] = {
  /**
   * Block for no worm condition.
   * @this Blockly.Block
   */
  init: function() {
    this.jsonInit({
      "message0": BlocklyGames.getMsg('Bird_noWorm'),
      "output": "Boolean",
      "colour": Bird.Blocks.VARIABLES_HUE,
      "tooltip": BlocklyGames.getMsg('Bird_noWormTooltip')
    });
  }
};

Blockly.JavaScript['bird_noWorm'] = function(block) {
  // Generate JavaScript for no worm condition.
  return ['noWorm()', Blockly.JavaScript.ORDER_FUNCTION_CALL];
};

Blockly.Blocks['bird_heading'] = {
```

```
    /**
     * Block for moving bird in a direction.
     * @this Blockly.Block
     */
    init: function() {
      this.setColour(Bird.Blocks.MOVEMENT_HUE);
      this.appendDummyInput()
          .appendField(BlocklyGames.getMsg('Bird_heading'))
          .appendField(new Blockly.FieldAngle('90'), 'ANGLE');
      this.setPreviousStatement(true);
      this.setTooltip(BlocklyGames.getMsg('Bird_headingTooltip'));
    }
  };

  Blockly.JavaScript['bird_heading'] = function(block) {
    // Generate JavaScript for moving bird in a direction.
    var dir = parseFloat(block.getFieldValue('ANGLE'));
    return 'heading(' + dir + ', \'block_id_' + block.id + '\');\n';
  };

  Blockly.Blocks['bird_position'] = {
    /**
     * Block for getting bird's x or y position.
     * @this Blockly.Block
     */
    init: function() {
      this.jsonInit({
        "message0": "%1",
        "args0": [
          {
            "type": "field_dropdown",
            "name": "XY",
            "options": [["x", "X"], ["y", "Y"]]
          }
        ],
        "output": "Number",
        "colour": Bird.Blocks.VARIABLES_HUE,
        "tooltip": BlocklyGames.getMsg('Bird_positionTooltip')
      });
    }
  };

  Blockly.JavaScript['bird_position'] = function(block) {
    // Generate JavaScript for getting bird's x or y position.
    var code = 'get' + block.getFieldValue('XY').charAt(0) + '()';
    return [code, Blockly.JavaScript.ORDER_FUNCTION_CALL];
  };

  Blockly.Blocks['bird_compare'] = {
    /**
     * Block for comparing bird's x or y position with a number.
     * @this Blockly.Block
     */
    init: function() {
      this.setHelpUrl(Blockly.Msg['LOGIC_COMPARE_HELPURL']);
      var OPERATORS = [['\u200F<', 'LT'], ['\u200F>', 'GT']];
      this.setColour(Blockly.Msg['LOGIC_HUE']);
      this.setOutput(true, 'Boolean');
      this.appendValueInput('A')
          .setCheck('Number');
      this.appendValueInput('B')
          .setCheck('Number')
          .appendField(new Blockly.FieldDropdown(OPERATORS), 'OP');
      this.setInputsInline(true);
      // Assign 'this' to a variable for use in the tooltip closure below.
      var thisBlock = this;
      this.setTooltip(function() {
        var op = thisBlock.getFieldValue('OP');
        var TOOLTIPS = {
```

```
          'LT': Blockly.Msg['LOGIC_COMPARE_TOOLTIP_LT'],
          'GT': Blockly.Msg['LOGIC_COMPARE_TOOLTIP_GT']
        };
        return TOOLTIPS[op];
      });
    }
  };

  Blockly.JavaScript['bird_compare'] = function(block) {
    // Generate JavaScript for comparing bird's x or y position with a number.
    var operator = (block.getFieldValue('OP') == 'LT') ? '<' : '>';
    var order = Blockly.JavaScript.ORDER_RELATIONAL;
    var argument0 = Blockly.JavaScript.valueToCode(block, 'A', order) || '0';
    var argument1 = Blockly.JavaScript.valueToCode(block, 'B', order) || '0';
    var code = argument0 + ' ' + operator + ' ' + argument1;
    return [code, order];
  };

  Blockly.Blocks['bird_and'] = {
    /**
     * Block for logical operator 'and'.
     * @this Blockly.Block
     */
    init: function() {
      this.setHelpUrl(Blockly.Msg['LOGIC_OPERATION_HELPURL']);
      this.setColour(Blockly.Msg['LOGIC_HUE']);
      this.setOutput(true, 'Boolean');
      this.appendValueInput('A')
          .setCheck('Boolean');
      this.appendValueInput('B')
          .setCheck('Boolean')
          .appendField(Blockly.Msg['LOGIC_OPERATION_AND']);
      this.setInputsInline(true);
      this.setTooltip(Blockly.Msg['LOGIC_OPERATION_TOOLTIP_AND']);
    }
  };

  Blockly.JavaScript['bird_and'] = function(block) {
    // Generate JavaScript for logical operator 'and'.
    var order = Blockly.JavaScript.ORDER_LOGICAL_AND;
    var argument0 = Blockly.JavaScript.valueToCode(block, 'A', order);
    var argument1 = Blockly.JavaScript.valueToCode(block, 'B', order);
    if (!argument0 && !argument1) {
      // If there are no arguments, then the return value is false.
      argument0 = 'false';
      argument1 = 'false';
    } else {
      // Single missing arguments have no effect on the return value.
      if (!argument0) {
        argument0 = 'true';
      }
      if (!argument1) {
        argument1 = 'true';
      }
    }
    var code = argument0 + ' && ' + argument1;
    return [code, order];
  };

  Blockly.Blocks['bird_ifElse'] = {
    /**
     * Block for 'if/else'.
     * @this Blockly.Block
     */
    init: function() {
      this.setHelpUrl(Blockly.Msg['CONTROLS_IF_HELPURL']);
      this.setColour(Blockly.Msg['LOGIC_HUE']);
      this.appendValueInput('CONDITION')
          .appendField(Blockly.Msg['CONTROLS_IF_MSG_IF'])
```

```
208           .setCheck('Boolean');
209       this.appendStatementInput('DO')
210           .appendField(Blockly.Msg['CONTROLS_IF_MSG_THEN']);
211       this.appendStatementInput('ELSE')
212           .appendField(Blockly.Msg['CONTROLS_IF_MSG_ELSE']);
213       this.setDeletable(false);
214       this.setTooltip(Blockly.Msg['CONTROLS_IF_TOOLTIP_2']);
215     }
216   };
217
218   Blockly.JavaScript['bird_ifElse'] = function(block) {
219     // Generate JavaScript for 'if/else' conditional.
220     var argument = Blockly.JavaScript.valueToCode(block, 'CONDITION',
221                    Blockly.JavaScript.ORDER_NONE) || 'false';
222     var branch0 = Blockly.JavaScript.statementToCode(block, 'DO');
223     var branch1 = Blockly.JavaScript.statementToCode(block, 'ELSE');
224     var code = 'if (' + argument + ') {\n' + branch0 +
225                '} else {\n' + branch1 + '}\n';
226     return code;
227   };
228
229   // Backup the initialization function on the stock 'if' block.
230   Blockly.Blocks['controls_if'].oldInit = Blockly.Blocks['controls_if'].init;
231
232     /**
233      * Modify the stock 'if' block to be a singleton.
234      * @this Blockly.Block
235      */
236   Blockly.Blocks['controls_if'].init = function() {
237     this.oldInit();
238     this.setPreviousStatement(false);
239     this.setNextStatement(false);
240     this.setDeletable(false);
241   };
242
```